**Recognize identifiers**

/\*lex code to determine whether input is an identifier or not\*/

% {

#include <stdio.h>

%}

/ rule section % %

// regex for valid identifiers

^[a - z A - Z \_][a - z A - Z 0 - 9 \_] \* printf("Valid Identifier");

// regex for invalid identifiers

^[^a - z A - Z \_] printf("Invalid Identifier");

.;

% %

main()

{

yylex();

}

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

void main()

{

char a[10];

int flag, i=1;

clrscr();

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

gets(a);

if(isalpha(a[0]))

flag=1;

else

printf("\n Not a valid identifier");

while(a[i]!='\0')

{

if(!isdigit(a[i])&&!isalpha(a[i]))

{

flag=0;

break;

}

i++;

}

if(flag==1)

printf("\n Valid identifier");

getch();

}

**Recognize language of string ending with “ab”**

//lex program

%{

#include <stdio.h>

%}

str [a-z A-Z]\*(ab)

%%

{str} printf("\n accepted");

.\* printf("\n rejected");

%%

main()

{

printf("Enter the string:");

yylex();

}

#recognize the lanuage of string endiing with ab

def switcher(state,l):

if state == 0:

if l == 'a':

return 1

elif l == 'b':

return 0

elif state == 1:

if l == 'a':

return 1

elif l == 'b':

return 2

elif state == 2:

if l == 'a' :

return 1

elif l == 'b':

return 0

state = 0

s = input("Enter the string: ")

for l in s:

state = switcher(state,l)

print (state)

if state == 2:

print("String is accepted")

else :

print("String is Rejected")

**Recognize language of string containing substring “bab”**

#recognize the lanuage of string with substring bab

def switcher(state,l):

if state == 0:

if l == 'a':

return 0

elif l == 'b':

return 1

elif state == 1:

if l == 'a':

return 2

elif l == 'b':

return 1

elif state == 2:

if l == 'a' :

return 0

elif l == 'b':

return 3

elif state == 3:

if l == 'a' or l == 'b' :

return 3

state = 0

s = input("Enter the string: ")

for l in s:

state = switcher(state,l)

print (state)

if state == 3:

print("String is accepted")

else :

print("String is Rejected")

class Main {

public static void main(String[] args) {

// create a string

String txt = "aeasdbablklk";

String str1 = "bab";

// check if name is present in txt

// using contains()

boolean result = txt.contains(str1);

if(result) {

System.out.println(str1 + " is present in the string.");

}

else {

System.out.println(str1 + " is not present in the string.");

}

}

}

**Recognize relational operators**

#include<stdio.h>

#include<conio.h>

void main()

{

char s[5];

printf("\n Enter any operator:");

gets(s);

switch(s[0])

{

case'>': if(s[1]=='>')

printf("\n Greater than");

break;

case'<': if(s[1]=='<')

printf("\nLess than");

break;

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

printf("\nEqual to");

break;

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

printf("\nNot Equal");

break;

case'&&': if(s[1]=='&&')

printf("\nLogical AND");

break;

case'||': if(s[1]=='||')

printf("\nLogical OR");

break;

default: printf("\n Not a operator");

}

getch();

}

#recognise the realtional operator

rel\_op = {

"==" : "Double Equals to",

">" : "greater than",

"<" : "less than",

"<=" : "less than equals to",

">=" : "Greater than Equals to",

}

inp = input("Enter the Relational operator: ")

if inp in rel\_op:

print(rel\_op[inp])

else :

print("No such operator")

**Count character, lines, spaces, numbers ..**

Countp.txt

your Name

This is a sample

123

&& bye

%{

int lines=0, words=0, capital=0, small=0, num=0, space=0, other=0;

%}

%%

\n {lines++;words++}

[\t'']words++;

[A-Z]capital++;

[a-z]small++;

[0-9]num++;

([ ])+ space++;

. {other++;}

%%

int main()

{

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

yylex();

printf("Lines=%d\n", lines);

printf("Words=%d\n",words);

printf("Small letters=%d\n", small);

printf("Capital letters=%d\n",capital);

printf("Numbers=%d\n", num);

printf("Spaces=%d\n", space);

printf("Other=%d\n", other);

return 0;

}

int yywrap()

{

return 1;

}

Calculator using Yacc

Calc.l

%{

   /\* Definition section \*/

  #include<stdio.h>

  #include "y.tab.h"

**extern** **int** yylval;

%}

/\* Rule Section \*/

%%

[0-9]+ {

          yylval=**atoi**(yytext);

**return** NUMBER;

       }

[\t] ;

[\n] **return** 0;

. **return** yytext[0];

%%

**int** yywrap()

{

**return** 1;

}

Calc.y

%{

/\* Definition section \*/

#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 which can have operations Addition, Subtraction, Multiplication, Division, Modulus and Round brackets:\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;

}

**Three address code**

#include<stdio.h>

#include<string.h>

void dm();

void as();

inti,j,l;

char ex[10],expr[10] ,expr1[10];

void main()

{

printf("\nEnter an Arithematic Expression: ");

scanf("%s",ex);

strcpy(expr,ex);

l=strlen(expr);

expr1[0]='\0';

for(i=0;i<l;i++)

{

if(expr[i+2]=='/'||expr[i+2]=='\*'){

if(expr[i]=='+'||expr[i]=='-') {

dm();

break; }

else{

as();

break; }}}}

void dm() {

strrev(expr);

j=l-i-1;

strncat(expr1,expr,j);

strrev(expr1);

printf("Three Address Code:\nt1=%s\nt2=%c%ct1\nx=t2\n",expr1,expr[j+1],expr[j]); }

void as() {

strncat(expr1,expr,i+2);

printf("Three Address Code:\nt1=%s\nt2=t1%c%c\nx=t2\n",expr1,expr[i+2],expr[i+3]);

}

**Symbol table for 2 pass assembler**

instructions=[[" ","START","200"," "],

[" ","MOVER","AREG,","DATA"],

[" ","MOVER","BREG,","=4"],

["X","EQU","10"," "],

[" ","LTORG"," "," "],

["DATA","DC","5"," "],

["ST","DS","10"," "],

[" ","MOVER","BREG,","=5"],

[" ","END"," "," "] ]

LocationCounter=[]

LC=int(instructions[0][2])

LocationCounter.append(LC)

for i in instructions[1:]:

LocationCounter.append(LC)

if(i[1]=="EQU"):

continue

elif(i[1]=="DS"):

LC+=int(i[2])

else:

LC+=1

#print(LocationCounter)

SymbolTable=[]

for i in range(len(instructions)):

temp=[]

if instructions[i][1]=="EQU":

temp.append(instructions[i][0])

temp.append(int(instructions[i][2]))

SymbolTable.append(temp)

elif instructions[i][0]==" " and instructions[i][3]!=" " and (not instructions[i][3].startswith("=")):

temp.append(instructions[i][3])

SymbolTable.append(temp)

elif instructions[i][0]!=" " and [instructions[i][0]] not in SymbolTable:

temp.append(instructions[i][0])

temp.append(LocationCounter[i])

SymbolTable.append(temp)

elif instructions[i][0]!=" " and [instructions[i][0]] in SymbolTable:

SymbolTable[SymbolTable.index([instructions[i][0]])].append(LocationCounter[i])

#print(SymbolTable)

print(" Symbol Table ")

print("---------------------------------")

count=0

length=1

print("|Index\t|Symbol\t|Address|Length\t|")

print("---------------------------------")

for i in SymbolTable:

print("|%d\t|%s\t|%d\t|%d\t|"%(count,i[0],i[1],length))

count+=1

print("---------------------------------")

**Literal table for 2 pass assembler**

lc=0

ins=["START 200","MOVER AREG,DATA","MOVER BREG,=4","X EQU

10","LTORG","DATA DC 5","ST DS 10","MOVER BREG,=5","END"]

inst,addr=ins[0].split(" ")

if(inst=="START"):

lc=int(addr)

litname=[]

litarr=[]

for i in range(1,len(ins)):

if "DS" in ins[i]:

num=int(ins[i].split(" ")[-1])

lc+=num

elif(ins[i]=="LTORG"):

lc-=1

litarr.append(lc)

lc+=1

elif(ins[i]=="END"):

litarr.append(lc)

else:

if "=" in ins[i]:

instr,lit=ins[i].split("=")

litname.append("="+lit)

lc+=1

print("\*\*\*\*\*Literal Table\*\*\*\*\*")

print("index\tname\taddress")

for i in range(len(litname)):

print(str(i)+"\t"+litname[i]+"\t"+str(litarr[i]))

**Intermediate code for two pass assembler**

**MDT, MNT, ALA for macro processor**

MTDC = 0

# file=open("in.txt")

# a = file.readlines()

a= ['MACRO', 'ADDM &arg1 &arg2 &arg3', 'A 1, &arg1', 'A 2, &arg2', 'A 3, &arg3', 'MEND']

#ALA Table

ala={}

ala\_statement = []

for i in a:

if "MACRO" in i:

MACRO = True

continue

if MACRO == True:

ala\_statement.append(i)

break

ala\_statement\_1= "".join(ala\_statement)

ala\_statement\_word = ala\_statement\_1.split()

count = 0

for i in ala\_statement\_word:

if count ==0:

ala['&lab'] = f'#{count}'

count +=1

else:

if '&' in i:

ala[i] = f'#{count}'

count +=1

#Macro Defination Table

ans = {}

MACRO = False

MEND = False

count\_i = 0

reg = None

key = None

for i in a:

if "MACRO" in i and MACRO == False and MEND == False:

MACRO = True

continue

if i != "MACRO" and MACRO == True and MEND == False:

if count\_i == 0:

ans[count\_i] = f'&lab {i}'

count\_i +=1

else:

a= i.split()

b = ala.keys()

c = []

for k in b:

continued

c.append(k)

for j in a:

if j in c:

reg = ala[j]

key = j

if reg!= None:

ans\_1= i.replace(key,reg)

if count\_i ==1:

ans[count\_i] = f'#0 {ans\_1}'

else:

ans[count\_i] = ans\_1

count\_i+=1

def print\_macro\_defination\_table():

print("\tMDT\t")

print("-"\*20)

print("Index\t Instruction")

for key,value in ans.items():

print(f'{key}\t{value}\t')

print("-"\*20)

def print\_ALA\_Table():

print("\tALA\t")

print("-"\*20)

print("Index\t Name")

for key,value in ala.items():

print(f'{value}\t{key}\t')

print("-"\*20)

print\_macro\_defination\_table()

print()

print()

print\_ALA\_Table()

code= [['MACRO'],

['&LAB','ADDM','&arg1','&arg2','&arg3'],

['&LAB','A','1','&arg1'],

['','A','2','&arg2'],

['','A','3','&arg3'],

['MEND']

]

DC = {'D1':4, 'D2':5, 'D3':6}

mCall = 'L1 ADDM D1 D2 D3'

callList = mCall.split(' ')

cList = list(callList)

callList.remove(code[1][1])

MDT = [['&LAB','ADDM','&arg1','&arg2','&arg3'],

['#0','A','1','#1'],

continued

['','A','2','#2'],

['','A','3','#3'],

['MEND']]

MNT = [[0,'ADDM',0]]

dummyALA = [['#0','&LAB'],

['#1','&arg1'],

['#2','&arg2'],

['#3','&arg3']]

expansionTable = []

actualALA = {}

x = 0

for i,j in enumerate(callList):

if code[1][1]==j:

continue

else:

actualALA[dummyALA[i][0]] = j

for i in MDT[1:-1]:

if len(expansionTable)==0:

expansionTable.append([cList[0], i[1], i[2], actualALA[i[3]]])

else:

expansionTable.append(['',i[1], i[2], actualALA[i[3]]])

print()

print('\tExpansion Table')

for i in expansionTable:

print('%s\t%s\t%s\t%s\t'%(i[0],i[1],i[2],i[3]))

**compute first for grammer**

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

char t[5],nt[10],p[5][5],first[5][5],temp;

int i,j,not,nont,k=0,f=0;

clrscr();

printf("\nEnter the no. of Non-terminals in the grammer:");

scanf("%d",&nont);

printf("\nEnter the Non-terminals in the grammer:\n");

for(i=0;i<nont;i++)

{

scanf("\n%c",&nt[i]);

}

printf("\nEnter the no. of Terminals in the grammer: ( Enter e for absiline ) ");

scanf("%d",&not);

printf("\nEnter the Terminals in the grammer:\n");

for(i=0;i<not||t[i]=='$';i++)

{

scanf("\n%c",&t[i]);

}

for(i=0;i<nont;i++)

{

p[i][0]=nt[i];

first[i][0]=nt[i];

}

printf("\nEnter the productions :\n");

for(i=0;i<nont;i++)

{

scanf("%c",&temp);

printf("\nEnter the production for %c ( End the production with '$' sign )

:",p[i][0]);

for(j=0;p[i][j]!='$';)

{

j+=1;

scanf("%c",&p[i][j]);

}

}

for(i=0;i<nont;i++)

{

printf("\nThe production for %c -> ",p[i][0]);

for(j=1;p[i][j]!='$';j++)

{

printf("%c",p[i][j]);

}

}

for(i=0;i<nont;i++)

{

f=0;

for(j=1;p[i][j]!='$';j++)

{

for(k=0;k<not;k++)

{

if(f==1)

break;

if(p[i][j]==t[k])

{

first[i][j]=t[k];

first[i][j+1]='$';

f=1;

break;

}

else if(p[i][j]==nt[k])

{

first[i][j]=first[k][j];

if(first[i][j]=='e')

continue;

first[i][j+1]='$';

f=1;

break;

}

}

}

}

for(i=0;i<nont;i++)

{

printf("\n\nThe first of %c -> ",first[i][0]);

for(j=1;first[i][j]!='$';j++)

{

printf("%c\t",first[i][j]);

}

}

getch();

}

**target code for compiler**

#include<stdio.h>

#include<conio.h>

#include<string.h>

char op[2], arg1[5], arg2[5], result[5];

void

main ()

{

  FILE \*fp1, \*fp2;

  fp1 = fopen ("input.txt", "r");

  fp2 = fopen ("output.txt", "w");

  while (!feof (fp1))

    {

      fscanf (fp1, "%s%s%s%s", op, arg1, arg2, result);

      if (strcmp (op, "+") == 0)

  {

  fprintf (fp2, "\nMOV R0,%s", arg1);

  fprintf (fp2, "\nADD R0,%s", arg2);

  }

      if (strcmp (op, "\*") == 0)

{

  fprintf (fp2, "\nMOV R0,%s", arg1);

  fprintf (fp2, "\nMUL R0,%s", arg2);

  fprintf (fp2, "\nMOV %s,R0", result);

}

      if (strcmp (op, "-") == 0)

{

  fprintf (fp2, "\nMOV R0,%s", arg1);

  fprintf (fp2, "\nSUB R0,%s", arg2);

  fprintf (fp2, "\nMOV %s,R0", result);

}

      if (strcmp (op, "/") == 0)

{

  fprintf (fp2, "\nMOV R0,%s", arg1);

  fprintf (fp2, "\nDIV R0,%s", arg2);

  fprintf (fp2, "\nMOV %s,R0", result);}

      if (strcmp (op, "=") == 0){

  fprintf (fp2, "\nMOV %s,R0", result);}

    }

  fclose (fp1);

  fclose (fp2);

  getch ();}

**Input.txt**

+ a b t1

= t1 ? x

**Output.txt**

MOV R0,a

ADD R0,b

MOV x,R0