Program to recognize identifiers

| import re  s = input("enter the string:")  pattern = r'^[a-zA-Z\_]\w\*$'  if(re.search(pattern,s)):  print("\n valid identifier")  else:  print("\n invalid identifier") |
| --- |

String ending with ab

| #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") |
| --- |

String containing substring as 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") |
| --- |

Program to recognize the relational operators

| #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") |
| --- |

Program to count no of characters lines tabs numbers and blank spaces

| Main.c file  #include<stdio.h>  #include<conio.h>  int main()  {  FILE \*fp;  char ch, fname[30];  int noOfChar=0, noOfSpace=0, noOfTab=0, noOfNewline=0, noOfDigits=0;  printf("Enter file name with extension: ");  gets(fname);  fp = fopen(fname, "r");  while(fp)  {  ch = fgetc(fp);  if(ch==EOF)  break;  noOfChar++;  if(ch==' ')  noOfSpace++;  if(ch=='\t')  noOfTab++;  if(ch=='\n')  noOfNewline++;  if(ch>='0' && ch<='9')  noOfDigits++;  }  fclose(fp);  printf("\nNumber of characters = %d", noOfChar);  printf("\nNumber of spaces = %d", noOfSpace);  printf("\nNumber of tabs = %d", noOfTab);  printf("\nNumber of newline = %d", noOfNewline);  printf("\nNumber of digits = %d", noOfDigits);  getch();  return 0;  }  Txt.txt file  HELLO I AM POOJA.  i am studying in RAIT.  89670. |
| --- |

Calculator using YACC

| **.y file**  %{  #include<stdio.h>  #include<math.h>  %}  %union {float num;}  %start line  %token cos1  %token sin1  %token tan1  %token <num> number  %type <num> exp  %%  line : exp  | line exp  ;  exp : number {$$=$1;}  | exp '+' number {$$=$1+$3;printf("\n%f+%f=%f\n",$1,$3,$$);}  | exp '-' number {$$=$1-$3;printf("\n%f-%f=%f\n",$1,$3,$$);}  | exp '\*' number {$$=$1\*$3;printf("\n%f\*%f=%f\n",$1,$3,$$);}  | exp '/' number {$$=$1/$3;printf("\n%f/%f=%f\n",$1,$3,$$);}  | cos1 number {printf("%f",cos(($2/180)\*3.14));}  | sin1 number {printf("%f",sin(($2/180)\*3.14));}  | tan1 number {printf("%f",tan(($2/180)\*3.14));}  ;  %%  int main(){  yyparse();  return 0;  }  int yyerror(){  //exit(0);  }  **.l file**  %{  #include "y.tab.h"  %}  %%  [0-9]+ {yylval.num=atof(yytext); return number;}  [-+\*/] {return yytext[0];}  COS|cos {return cos1; }  SIN|sin {return sin1; }  TAN|tan {return tan1; }  %%  int yywrap(){  return 1;  }  Flex commands  bison –dy expt3.y  flex expt3.l  gcc lex.yy.c y.tab.c  a.exe |
| --- |

Symbol table of two pass assembler

| #include <bits/stdc++.h>  #include <vector>  #include <string.h>  using namespace std;  int main()  {  int c1 = 0, sr = 0;  vector<int> lc,lit1, lit2, lit3;  vector<vector<string>> sym;  vector<vector<string>> v{  {"", "START", "200", " "},  {"", "MOVER", "AREG", "DATA"},  {"", "MOVER", "BREG", "=1"},  {"", "ADD", "AREG", "BREG"},  {"", "LTORG", " ", " "},  {"DATA", "DC", "5", " "},  {"ST ", "DS", "10", " "},  {"", "ORIGIN", "ST+20", " "},  {"", "ADD", "CREG", "=2"},  {"", "END", " ", " "}};  int num = stoi(v[0][2]), ST = 205;  if (v[0][1] == "START")  {  lc.push\_back(num);  }for (int i = 0; i < v.size(); i++)  {  for (int j = 0; j < v[i].size(); j++)  {  cout << v[i][j] << "\t ";  }  if(v[i][1]=="LTORG" || v[i][1]=="END")  {  lit3.push\_back(num); }    if (v[i][1] == "MOVER" || v[i][1] == "ADD" || v[i][1] == "LTORG" || v[i][1] == "DC" || v[i][1] == "END")  {  lc.push\_back(num);  num++;  }  if (v[i][1] == "DS")  {  lc.push\_back(num);  num += stoi(v[i][2]);  }  if (v[i][1] == "ORIGIN")  {  lc.push\_back(num);  string a = (v[i][2]).substr(3, 5);  num = ST + stoi(a);  }  cout << lc[i];  cout << endl;  }    //symbol:    cout << "--- SYMBOL TABLE ---" << endl;  cout << "\tname\taddress\tlength" << endl;  for (int i = 0; i < v.size(); i++)  {  string a = (v[i][0]);  if (a.size() != 0)  {  cout << i -5 << "\t";  cout << a << "\t";  if (a == "DATA")  {  cout << lc[i] << "\t";  }  else  {  cout << lc[i] << "\t";  }  cout << "1\t";  cout << endl;  }  }  for (int i = 0; i < sym.size(); i++)  {  for (int j = 0; j < sym[i].size(); j++)  {  cout << sym[i][j] << " ";  }  cout << endl;  }      for (int i = 0; i < sym.size(); i++)  {  for (int j = 0; j < sym[i].size(); j++)  {  cout << sym[i][j] << " ";  }  cout << endl;  }    return 0;  } |
| --- |

Literal table

| #include <bits/stdc++.h>  #include <vector>  #include <string.h>  using namespace std;  int main()  {  int c1 = 0, sr = 0;  vector<int> lc,lit1, lit2, lit3;  vector<vector<string>> sym;  vector<vector<string>> v{  {"", "START", "200", " "},  {"", "MOVER", "AREG", "DATA"},  {"", "MOVER", "BREG", "=1"},  {"", "ADD", "AREG", "BREG"},  {"", "LTORG", " ", " "},  {"DATA", "DC", "5", " "},  {"ST ", "DS", "10", " "},  {"", "ORIGIN", "ST+20", " "},  {"", "ADD", "CREG", "=2"},  {"", "END", " ", " "}};  int num = stoi(v[0][2]), ST = 205;  if (v[0][1] == "START")  {  lc.push\_back(num);  }for (int i = 0; i < v.size(); i++)  {  for (int j = 0; j < v[i].size(); j++)  {  cout << v[i][j] << "\t ";  }  if(v[i][1]=="LTORG" || v[i][1]=="END")  {  lit3.push\_back(num); }    if (v[i][1] == "MOVER" || v[i][1] == "ADD" || v[i][1] == "LTORG" || v[i][1] == "DC" || v[i][1] == "END")  {  lc.push\_back(num);  num++;  }  if (v[i][1] == "DS")  {  lc.push\_back(num);  num += stoi(v[i][2]);  }  if (v[i][1] == "ORIGIN")  {  lc.push\_back(num);  string a = (v[i][2]).substr(3, 5);  num = ST + stoi(a);  }  cout << lc[i];  cout << endl;  }  //literal:    for(int i=0;i<v.size();i++)  {  string a =(v[i][3]).substr(0,1);    if(a == "=")  {  string c =(v[i][3]).substr(1,2);  int tmp1 = stoi(c);    lit1.push\_back(sr);  sr+=1;    lit2.push\_back(tmp1);  }  }    if(c1==0)  {  cout<<endl<<"Literal Table:"<<endl;  cout<<"Sr. No "<<"\tName "<<"\tAddress"<<endl;  c1+=1;  }    for(int i = 0; i < 2; i++)  {  cout<<lit1[i]<<"\t ";  cout<<lit2[i]<<"\t ";  cout<<lit3[i]<<"\t ";  cout<<endl;  }return 0;  } |
| --- |

Intermediate code

| symbol = {  'B': [0, 203, 1],  'A': [1, 204, 1],  'X': [2, 202, 1],  'L': [3, 205, 1],  'C': [4, 7, 1]  }  literal = {  '=3': 0,  '=4': 1  }  registers = {  'AREG': '01',  'BREG': '02',  'CREG': '03'  }  program = [  ('', 'START', '200', ''),  ('', 'MOVER', 'AREG', 'B'),  ('', 'MOVER', 'BREG', 'A'),  ('X', 'ADD', 'AREG', '=3'),  ('B', 'DC', '5', ''),  ('A', 'DC', '4', ''),  ('L', 'DS', '2', ''),  ('', 'ADD', 'BREG', '=4'),  ('', 'LTORG', '0', ''),  ('', 'ADD', 'CREG', 'C'),  ('C', 'EQU', '7', ''),  ('', 'END', '0', '')  ]  operand = {  'IS': [('MOVER', '04'), ('ADD', '02')],  'AD': [('START', '00'), ('END', '02'), ('LTORG', '05'), ('EQU', '03')],  'DL': [('DC', '02'), ('DS', '01')]  }  for statement in program:  label, inst, op1, op2 = statement  curr\_code = ''  if op1[0] == '=':  type1 = 'L'  elif op1 in ['AREG', 'BREG', 'CREG']:  type1 = 'RG'  op1 = registers[op1]  elif op1.isdigit():  type1 = 'C'  else:  type1 = 'S'  op1 = symbol[op1][0]  type2 = ''  if op2:  if op2[0] == '=':  type2 = 'L'  op2 = literal[op2]  elif op2 in ['AREG', 'BREG', 'CREG']:  type2 = 'RG'  op2 = registers[op2]  elif op2.isdigit():  type2 = 'C'  elif op2.isalpha():  type2 = 'S'  op2 = symbol[op2][0]  if label:  curr\_code += f'(S, {symbol[label][0]})'  for key, value in operand.items():  for i in value:  if i[0] == inst:  curr\_code += f'({key}, {i[1]}) ({type1}, {op1})'  if op2:  curr\_code += f'({type2}, {op2})'  print(curr\_code) |
| --- |

3 address code

| precedence = {'+':1, '-':1, '\*':2,'/':2,'^':3}  op = {'+','-','\*','/'}  X = input("Enter an infix arithmetic expression\n")  stack = []  postfix=""  for i in X:  if (i in op):  if len(stack)==0:  stack.append(i)  elif len(stack)!=0:  if (precedence[i]>precedence[stack[-1]]):  stack.append(i)  else:  while(len(stack)!=0 and (precedence[i] <= precedence[stack[-1]])):  t = stack.pop()  postfix+=t  stack.append(i)    else :  postfix+=i    while stack:  postfix+=stack.pop()  print(postfix)  #for converting to 3-address code  counter = 1  expstack = []  for i in postfix:  if i not in op:  expstack.append(i)  else:  a = expstack.pop()  b = expstack.pop()  print("t"+str(counter) + " = " + b + " "+ i + " "+ a)  expstack.append("t"+str(counter))  counter+=1  print("x = t"+str(counter-1)) |
| --- |

Target code

| # th\_cod\_gen = ['t1=a-b' , 't2=a-c' , 't3=t1+u','t4=t3+t2']  th\_cod\_gen = ['t1=a\*b' , 't2=d\*c' ,'t3=t1-t2']  # th\_cod\_gen = ['t1=c\*d','t2=t1+b']  out =[]  opval = {'+':'ADD' , '-':'SUB' , '\*':'MUL' , '/':'DIV'}  op = ['\*' , '+' , '-' , '/']  j=1  temp = []  for i in range(len(th\_cod\_gen)):  if len(th\_cod\_gen[i]) >= 6:  if th\_cod\_gen[i][2:].count('t') >= 1:  cnt = th\_cod\_gen[i][2:].count('t')  if cnt == 1:  opr = th\_cod\_gen[i][5]  tval = th\_cod\_gen[i][4]  out.append(f"{opval[opr]} {temp[int(tval) - 1]} , {th\_cod\_gen[i][6]}")  temp.append(f"R{int(tval)}")  else:  opr = th\_cod\_gen[i][5]  tval = th\_cod\_gen[i][4]  t2val = th\_cod\_gen[i][7]  out.append(f"{opval[opr]} {temp[int(tval) - 1]} , {temp[int(t2val) - 1]}")  else:  out.append(f"MOV {th\_cod\_gen[i][3]} , R{j}")  out.append(f"{opval[th\_cod\_gen[i][4]]} R{j} , {th\_cod\_gen[i][5]}")  temp.append(f"R{j}")  j+=1  print(\*out,sep='\n') |
| --- |

Calculate no of spaces

.l file

| %{  #include<stdio.h>  int lines=0, words=0,s\_letters=0,c\_letters=0, num=0, spl\_char=0,total=0;  %}  %%  \n { lines++; words++;}  [\t ' '] words++;  [A-Z] c\_letters++;  [a-z] s\_letters++;  [0-9] num++;  . spl\_char++;  %%  int main(void)  {  yyin= fopen("info.txt","r");  yylex();  total=s\_letters+c\_letters+num+spl\_char;  printf("\n This File contains ...");  printf("\n lines : %d", lines);  printf("\n words : %d", words);  printf("\n small letters : %d", s\_letters);  printf("\n capital letters : %d",c\_letters);  printf("\n digits : %d", num);  printf("\n special characters : %d",spl\_char);  printf("\n In total %d characters.\n",total);  }  int yywrap()  {  return(1);  } |
| --- |

Info.txt

I am Pooja i am studying in RAIT DyPatil@2001.

C:\SPCC\_19CE1031>cd\

C:\>cd C:\SPCC\_19CE1031

C:\SPCC\_19CE1031>flex Expt2.l

C:\SPCC\_19CE1031>gcc lex.yy.c

C:\SPCC\_19CE1031>a.exe

mdt ala

| code = ['MACRO',  '&Lab ADDM &arg1 &arg2 &arg3',  '&Lab A 1 &arg1',  'A 2 &arg2',  'A 3 &arg3',  'MEND'  ]  MNT = []  ALA = []  MDT = []  helper\_ala = []  MDTC = 0  for i, line in enumerate(code):  if line == 'MACRO':  continue  line\_arr = line.split()  if i == 1:  name = line\_arr[1]  MNT.append([len(MNT), name, MDTC])  index = 0  for arg in line\_arr:  if arg == name:  pass  else:  ALA.append([f'#{index}', arg])  helper\_ala.append(arg)  index += 1  MDT.append([len(MDT), line])  else:  entry = ''  for field in line\_arr:  if field in helper\_ala:  entry += '#' + str(helper\_ala.index(field)) + ' '  else:  entry += field + ' '  MDT.append([len(MDT), entry])  print('MNT:' + '\n')  for \_ in MNT:  print(\_)  print('\n')  print('ALA:' + '\n')  for \_ in ALA:  print(\_)  print('\n')  print('MDT:' + '\n')  for \_ in MDT:  print(\_)  print('\n') |
| --- |

first

| grammar\_inputs = ['S=aABc', 'A=b', 'B=#']    first = 'ab#'  follow = '##c'  non\_terminals=[]  terminals=[]    #extracting terminlas and non terminals    for i in range(len(grammar\_inputs)):  for j in grammar\_inputs[i]:  if j.isupper() and j not in non\_terminals:  non\_terminals.append(j)  elif (j.islower() or j=='#') and j not in terminals:  terminals.append(j)    #printing op  for i in range(len(non\_terminals)):  for j in range(len(terminals)):  if first[i] == terminals[j]:  if terminals [j] == '#':  print(f"[{non\_terminals[i]}, {follow[non\_terminals.index(non\_terminals[i])]}]= ({non\_terminals[i]}=>{grammar\_inputs[i][2:]})")  else:  print(f"[{non\_terminals[i]}, {terminals[j]}]=({non\_terminals[i]} => {grammar\_inputs[i][2:]})") |
| --- |