- **Recognise identifiers**

file-Prac1.l:

```
%{

%}

%%
^[a-zA-Z_][a-zA-Z0-9_] {printf("valid");}
^[^a-zA-Z_] {printf("invalid");}
. ;
%%

int main()
{
        yylex();
        return 0;
}

int yywrap(){
return 1;
}
```

RUNNING: flex prac1.l
Gcc lex.yy.c
A.exe

- **String ending with “ab”**

```
#include<stdio.h>
int main()
{
   int i=0 , state=0;
   char s[100];
   gets(s);
   while(s[i] !='\0')
        {
   switch(state){
        case 0: if(s[i]=='a') state=1;
              else if(s[i]=='b') state=2;
              else state=0;
        break;

        case 1: if(s[i]=='a') state=1;
```

```
            else if(s[i]=='b') state=3;
                else state=0;
            break;

            case 2: if(s[i]=='a') state=1;
                else if(s[i]=='b') state=2;
                else state=0;
            break;

            case 3: if(s[i]=='a') state=1;
                else if(s[i]=='b') state=2;
                else state=0;
            break;

          }
          i++;
      }
   if(state==3)
     printf("Accepted");
   else
     printf("Rejected");
   return 0;
 }
```

- **String containing “bab”**

```
#include<stdio.h>
int main()
{
    int i=0 , state=0;
    char s[100];
    gets(s);
    while(s[i] !='\0')
         {
    switch(state){
            case 0: if(s[i]=='a') state=1;
                else if(s[i]=='b') state=2;
                else state=0;
            break;

            case 1: if(s[i]=='a') state=1;
                else if(s[i]=='b') state=2;
                else state=0;
            break;
```

```
            case 2: if(s[i]=='a') state=3;
                    else if(s[i]=='b') state=2;
                    else state=0;
            break;

            case 3: if(s[i]=='a') state=1;
                    else if(s[i]=='b') state=4;
                    else state=0;
            break;

            case 4: if(s[i]=='a') state=4;
                    else if(s[i]=='b') state=4;
                    else state=0;
            break;
          }
          i++;
      }
   if(state==4)
     printf("Accepted");
   else
     printf("Rejected");
   return 0;
 }
```

- **Recognise relational operator**

```
#include <iostream>
using namespace std;
int main()
{
  int i=0 , state=0, flag=0; char s[100];
  cin>>s;
  while(s[i] !='\0')
  {
    switch(state){
      case 0: if(s[i]=='<') state=1;
              else if(s[i]=='>') state=5;
              else if(s[i]== '=') state=4;
              else state= 3;
              break;
      case 1: if(s[i] == '=') state=2;
              else if(s[i]=='>') state=9;
              break;
      case 5: if(s[i] == '=') state=6;
              else state=7;
```

```
            break;
      case 4: if(s[i]=='=')   state=8;
            break;
    }
    i++;
  }
  if(state==1) cout<<"L";
  if(state==2) cout<<"LE";
  if(state==3 || state==7) cout<<"Invalid";
  if(state==6) cout<<"GE";
  if(state==5) cout<<"G";
  if(state==8) cout<<"E";
  if(state==9)
  cout<<"NT";
   cout<<endl;
  return 0;
}
```

- **Count number of characters**

File-prac5.l

```
%{
int characters=0, numbers=0, tabs=0, lines=0, spaces=0;
%}

%%
[a-zA-Z] characters++;
[0-9]+ numbers++;
\n lines++;
\t tabs++;
" " spaces++;
%%

int main(){
yyin=fopen("demo.txt","r");
yylex();
printf("characters: %d\n",characters);
printf("lines: %d\n",lines);
printf("tabs: %d\n",tabs);
printf("numbers: %d\n",numbers);
printf("spaces: %d\n",spaces);
return 0;
}

int yywrap(){
```

```
return 1;
}
```

File-demo.txt-any text

- **YACC calculator**

File: prac6.l

```
%{
        #include "y.tab.h"
        extern int yylval;
%}

%%

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

\n   {return 0;}

.    {return yytext[0];}

%%
```

File: prac6.y

```
%{
        #include <stdio.h>
        #include <math.h>
        #include <stdlib.h>
%}
%token NUM
%left '+' '-' '*' '/'

%%
stmt : exp   {printf("\n Answer: %d \n", $1);}
        ;
exp :   exp '+' exp {$$=$1+$3;}
        | exp '-' exp {$$=$1-$3;}
        | exp '*' exp {$$=$1*$3;}
        | exp '/' exp {$$=$1/$3;}
        | NUM  {$$=$1;}
        ;
%%
```

```
int main()
{
        printf("Enter the arithmetic expression \n");
        yyparse();
        printf("Valid expression! ");
        return 0;

}
yyerror()
{
 printf("Invalid expression!");
 exit(0);
}

int yywrap()
{
 return 1;
}
```

Running: flex prac6.l
Bison -dy prac6.y
Gcc lex.yy.c y.tab.c
A.exe

- **Symbol table, literal table and intermediate code for 2 pass assembler**

```
ip=[['','START','200',''],
   ['','MOVR','AREG','DATA'],
   ['','MOVR','BREG','=4'],
   ['X','EQU','10',''],
   ['','LTORG','',''],
   ['DATA','DC','4',''],
   ['ST','DS','10',''],
   ['','MOVR','CREG','=5'],
   ['','END','','']]
instruction=['LTORG','MOVR','DC']
instruction1=['START','EQU','END','DS']
reg=['AREG','BREG','CREG']
lc=[]
lt=0

for z in ip:
    s=z[1]
    if s=='START':
        lt=int(z[2])
        lc.append(lt)
```

```
    elif s=='DS':
        lt=lt+int(z[2])
    elif s in instruction:
        lt=lt+1
    elif s=='END':
        break
    lc.append(lt)

st={}
for x in range(len(ip)):
    tempx=ip[x]
    for yn in range (len(tempx)):
        y=tempx[yn]
        if y not in reg and y not in instruction and y not in instruction1 and y.isalpha():
            if y in st.keys():
                st.update({y:lc[x]})
            elif yn==3:
                st[y]= '-'
            elif tempx[1]=='EQU':
                st[y] = tempx[2]
            elif yn==0:
                st[y]=lc[x]

literal={}
for x in range(len(ip)):
    if 'LTORG' in ip[x]:
        lit=x

for x in range(len(ip)):
    tempx=ip[x]
    if tempx[3]:
        k = tempx[3]
        if k[0]=='=':
            if x<lit:
                literal[k]=lc[lit]
            else:
                literal[k]=lc[-1]
                lc.append(lc[-1]+1)

print('symbol table')
index=0
for n in st:
    print(index, n, '\t', st[n], '\t', '1')
    index+=1
```

```
print()
print('literal table')
index=0
for n in literal:
    print(index, n, '\t', literal[n])
    index+=1
print()
print(lc)
```

```
symbol table
0 DATA   203     1
1 X      10      1
2 ST     204     1

literal table
0 =4     202
1 =5     215

[200, 200, 201, 202, 202, 203, 204, 214, 215, 216]
```

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

```
inp=[['MACRO'],
['&LAB', 'ADDM', '&ARG1', '&ARG2', '&ARG3'],
['&LAB', 'A', '1', '&ARG1'],
['','A', '2', '&ARG2'],
['','A', '3', '&ARG3'],
['MEND']]

inp1=[['MACRO'],
['&LAB', 'ADDM', '&ARG1', '&ARG2', '&ARG3'],
['&LAB', 'A', '1', '&ARG1'],
['','A', '2', '&ARG2'],
['','A', '3', '&ARG3'],
['MEND']]

op=['A', 'S']
mdtc=1
mntc=1
mdt=[]
mnt=[]
ala=[]

print ("MACRO definition table")
print('='*50)

def createala(xt):
    tc = xt
    tc.pop(1)
```

```
    ala.extend(tc)

for x in inp:
    if x[0]=='MACRO':
        continue
    elif len(x)>1 and x[1] not in op:
        print ([mdtc,x])
        mdtc+=1
        createala(x)
    elif len(x)>1 and x[1] in op :
        temp = x
        yt=[]
        for y in range (len(temp)):
            if temp[y] in ala :
                yt.append('#' + str(ala.index(temp[y])))
            else :
                yt.append(temp[y])
        t=[mdtc,yt]
        mdtc+=1
        mdt.append(t)
    elif x[0]=='MEND':
        t=[mdtc, ['MEND']]
        mdtc+=1
        mdt.append(t)



for d in mdt:
    print (d)

print()
print ("Arguement List Array")
print('='*50)

for i in range(len(ala)):
    print('#'+str(i),ala[i])

mdtc=1
for x in range(len(inp)):
    yt=[]
    lt=inp1[x]
    if lt[0]=='MACRO':
        yt.append(mntc)
        mntc+=1
        yk=inp1[x+1]
```

```
            print(yk)
            yt.append(yk[1])
            yt.append(mdtc)
            mdtc+=1
        else:
            mdtc+=1
        mnt.append(yt)

    print()
    print ("MACRO name table")
    print('='*50)

    for n in mnt:
        if n:
            print(n)
```

```
MACRO definition table
==================================================
[1, ['&LAB', 'ADDM', '&ARG1', '&ARG2', '&ARG3']]
[2, ['#0', 'A', '1', '#1']]
[3, ['', 'A', '2', '#2']]
[4, ['', 'A', '3', '#3']]
[5, ['MEND']]

Arguement List Array
==================================================
#0 &LAB
#1 &ARG1
#2 &ARG2
#3 &ARG3

MACRO name table
==================================================
[1, 'ADDM', 1]
```

- **FIRST for given grammar**

```
import sys
sys.setrecursionlimit(60)

def first(string):
    first_ = set()
    if string in non_terminals:
        alternatives = productions_dict[string]
        for alternative in alternatives:
            first_2 = first(alternative)
            first_ = first_ |first_2
    elif string in terminals:
        first_ = {string}
    elif string=='' or string=='@':
        first_ = {'@'}
```

```
    else:
        first_2 = first(string[0])
        if '@' in first_2:
            i = 1
            while '@' in first_2:
                first_ = first_ | (first_2 - {'@'})
                if string[i:] in terminals:
                    first_ = first_ | {string[i:]}
                    break
                elif string[i:] == '':
                    first_ = first_ | {'@'}
                    break
                first_2 = first(string[i:])
                first_ = first_ | first_2 - {'@'}
                i += 1
        else:
            first_ = first_ | first_2
    return  first_

no_of_terminals=int(input("Enter no. of terminals: "))

terminals = []

print("Enter the terminals :")
for _ in range(no_of_terminals):
    terminals.append(input())

no_of_non_terminals=int(input("Enter no. of non terminals: "))

non_terminals = []

print("Enter the non terminals :")
for _ in range(no_of_non_terminals):
    non_terminals.append(input())

starting_symbol = input("Enter the starting symbol: ")
no_of_productions = int(input("Enter no of productions: "))
productions = []

print("Enter the productions:")
for _ in range(no_of_productions):
    productions.append(input())
```

```
productions_dict = {}

for nT in non_terminals:
    productions_dict[nT] = []

for production in productions:
    nonterm_to_prod = production.split("->")
    alternatives = nonterm_to_prod[1].split("/")
    for alternative in alternatives:
        productions_dict[nonterm_to_prod[0]].append(alternative)

FIRST = {}

for non_terminal in non_terminals:
    FIRST[non_terminal] = set()
for non_terminal in non_terminals:
    FIRST[non_terminal] = FIRST[non_terminal] | first(non_terminal)

print("{: ^20}{: ^20}".format('Non Terminals','First'))
for non_terminal in non_terminals:
    print("{: ^20}{: ^20}".format(non_terminal,str(FIRST[non_terminal])))
```

```
Enter no. of terminals: 3
Enter the terminals :
a
b
c
Enter no. of non terminals: 3
Enter the non terminals :
S
A
B
Enter the starting symbol: S
Enter no of productions: 4
Enter the productions:
S->AaAb
S->BbBa
A->@
B->@
   Non Terminals           First
         S               {'b', 'a'}
         A                 {'@'}
         B                 {'@'}
```

- **3 address code**

```
exp = str(input())
post=[]
preced={'+':1, '-':1, "*":2, '/':2}
s=[]
```

```
for x in exp:
    if x.isalpha():
        post.append(x)
    elif not s or preced[x]>preced[s[-1]]:
        s.append(x)
    else:
        y=s.pop()
        post.append(y)
        s.append(x)
rs=s[::1]
post.extend(rs)
print(post)
es=[]
i=1

for z in post:
    if z.isalpha():
        es.append(z)
    else:
        r=es.pop()
        l=es.pop()
        print('t',i,'=',l,z,r)
        es.append('t'+str(i))
        i+=1
print('x = t',i-1)
```

```
a*b-c+d
['a', 'b', '*', 'c', '-', 'd', '+']
t 1 = a * b
t 2 = t1 - c
t 3 = t2 + d
x = t 3
```

- **Target code for compiler**

```
n = int(input("No. of lines :"))
tac =[]

def op(y):
    if y=='+':
        return 'ADD'
    elif y=='-':
        return 'SUB'
    elif y=='*':
        return 'MUL'
```

```
    elif y=='/':
        return 'DIV'

for _ in range(n):
    temp = str(input())
    tac.append(temp)
print(tac)
t=0
reg=[]
s=[]

for x in tac :
    if len(x)==5:
        if not s or s[-1]!=x[2]:
            reg.append('R'+str(t))
            s.append(x[2])
            print('MOV',reg[-1],s[-1])
            t+=1
            reg.append('R'+str(t))
            s.append(x[4])
            print('MOV',reg[-1],s[-1])
            t+=1
            print(op(x[3]),reg[-2],reg[-1])
            reg.append(reg[-2])
            s.append(x[0])

        elif s[-1]==x[2]:
            reg.append('R'+str(t))
            s.append(x[4])
            print('MOV',reg[-1],s[-1])
            t+=1
            print(op(x[3]),reg[-2],reg[-1])
            reg.append(reg[-2])
            s.append(x[0])

    elif len(x)==3:
        print('MOV',x[0],reg[-1])
```

```
No. of lines :4
1=a+b
2=1-c
3=2*d
x=3
['1=a+b', '2=1-c', '3=2*d', 'x=3']
MOV R0 a
MOV R1 b
ADD R0 R1
MOV R2 c
SUB R0 R2
MOV R3 d
MUL R0 R3
MOV x R0
```