1. Write a program to count characters, words, sentences, lines, tabs, numbers and blank spaces present in input using LEX (Input File: Text File)

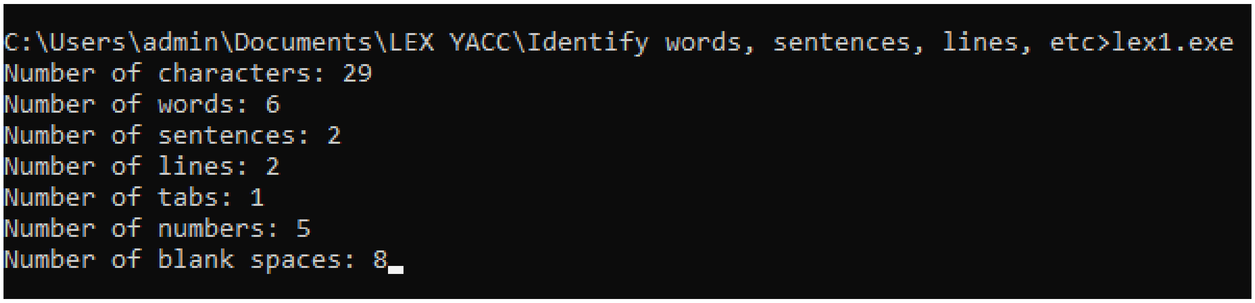
Program:

```
%option noyywrap
%{
     #include<stdio.h>
     int character =0;
     int word=0;
     int sentence=0;
     int line=0;
     int tab=0;
     int number=0;
     int space=0;
%}

%%
[' '] {space++;}
[\t] {tab++;}
[\n] {line++;}
[0-9]+ {number++;}
[a-zA-Z0-9]+ {++word; character += yyleng;}
[.]+ {sentence++;}

%%
int main()
{
FILE*ptr;
ptr=fopen("Test.txt","r");
yyin=ptr;
yylex();
printf("Number of characters: %d",character);
printf("\nNumber of words: %d",word);
printf("\nNumber of sentences: %d",sentence);
printf("\nNumber of lines: %d",line);
printf("\nNumber of tabs: %d",tab);
printf("\nNumber of numbers: %d",number);
printf("\nNumber of blank spaces: %d",space);
getch();
return 0;
}
```

sample.txt file:

```
Hello and Welcome.
This is Siddhesh.
        1 2 3 4 5
```

Output:



```
C:\Users\admin\Documents\LEX YACC\Identify words, sentences, lines, etc>lex1.exe
Number of characters: 29
Number of words: 6
Number of sentences: 2
Number of lines: 2
Number of tabs: 1
Number of numbers: 5
Number of blank spaces: 8
```

2. Write a program to recognize valid arithmetic expression that uses operator +,-,/,* using LEX and YACC

Program:

Calc.l

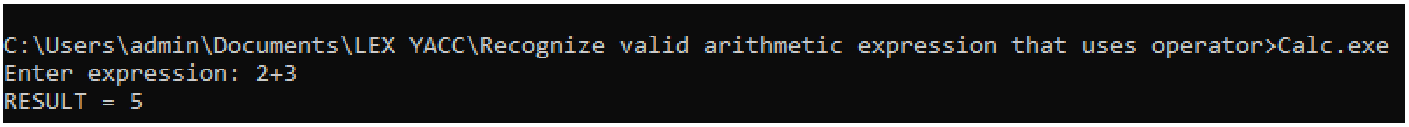
```
%option noyywrap
%{
      #include<math.h>
      #include"y.tab.h"
%}

%%
[0-9]+ {yylval=atoi(yytext); return NUM;}
[+-*/] {return *yytext;}
\n {return 0;}
. {return yytext[0];}
sin {return SIN;}
cos {return COS;}
tan {return TAN;}
log {return LOG;}

%%
```

Calc.y

```
%{
      #include<stdio.h>
      #include<math.h>
      #include<stdlib.h>
      int yylex(void);
      void yyerror(char *);
%}

%token NUM
%token SIN COS TAN LOG
%left '+''-'
%left '*''/'
%start s

%%
s     : exp {printf("RESULT = %d", $$);}
      ;
exp   : exp '+' exp {$$ = $1 + $3;}
      | exp '-' exp {$$ = $1 - $3;}
      | exp '*' exp {$$ = $1 * $3;}
      | exp '/' exp {
           if ($3 == 0) {
                printf("Division by zero");
                getch();
                exit(0);
           } else
                $$ = $1 / $3;
      }
      | SIN '('exp')' {$$ = sin($3);}
      | COS '('exp')' {$$ = cos($3);}
      | TAN '('exp')' {$$ = tan($3);}
      | LOG '('exp')' {$$ = log10($3);}
      | NUM {$$ = $1;}
      ;
%%

void yyerror(char *s){
      fprintf(stderr, "%s", s);
}

int main()
{
      printf("Enter expression: ");
      yyparse();
```

```
     getch();
     return 0;
}
```

Output:



```
C:\Users\admin\Documents\LEX YACC\Recognize valid arithmetic expression that uses operator>Calc.exe
Enter expression: 2+3
RESULT = 5
```

3. Write a program to recognize keywords/identifiers in C (Input File: C File)

Program:

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <stdlib.h>

bool isDelimiter(char ch){
   if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||  ch
== '/' || ch == ',' || ch == ';' || ch == '>' || ch == '<' ||
ch == '=' || ch == '(' || ch == ')' || ch == '[' || ch == ']'
|| ch == '{' || ch == '}')
    return (true);
   return (false);
}

bool isOperator(char ch){
   if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch
== '>' || ch== '<' || ch == '=')
    return (true);
   return (false);
}

bool isIdentifier(char* str){
   if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
str[0] == '3' || str[0] == '4' || str[0] == '5' || str[0] ==
'6' || str[0] == '7' || str[0] == '8' || str[0] == '9' ||
isDelimiter(str[0]) == true)
    return (false);
   return (true);
}
```

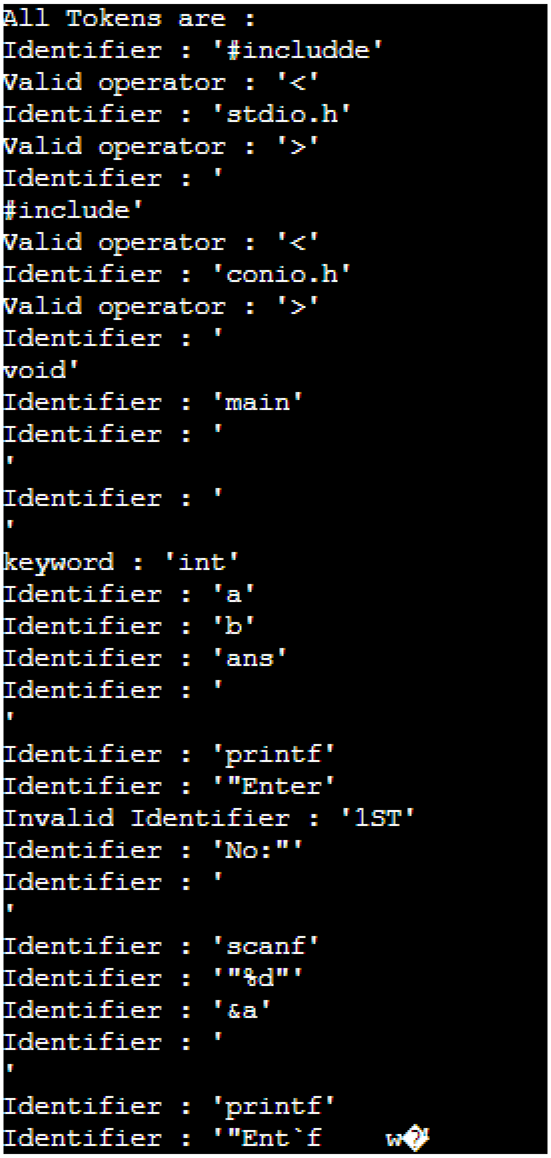
```
bool isKeyword(char* str) {
   if (!strcmp(str, "if") || !strcmp(str, "else") ||
!strcmp(str, "while") || !strcmp(str, "do") ||    !strcmp(str,
"break") || !strcmp(str, "continue") || !strcmp(str, "int") ||
!strcmp(str, "double") || !strcmp(str, "float") || !strcmp(str,
"return") || !strcmp(str, "char") || !strcmp(str, "case") ||
!strcmp(str, "char") || !strcmp(str, "sizeof") || !strcmp(str,
"long") || !strcmp(str, "short") || !strcmp(str, "typedef") ||
!strcmp(str, "switch") || !strcmp(str, "unsigned") ||
!strcmp(str, "void") || !strcmp(str, "static") || !strcmp(str,
"struct") || !strcmp(str, "goto"))
    return (true);
   return (false);
}

bool isInteger(char* str) {
   int i, len = strlen(str);
   bool hasDecimal = false;
   if (len == 0)
   return (false);
   for (i = 0; i < len; i++) {
       if (str[i] != '0' && str[i] != '1' && str[i] != '2' &&
str[i] != '3' && str[i] != '4' && str[i] != '5' && str[i] !=
'6' && str[i] != '7' && str[i] != '8' && str[i] != '9' &&
str[i] != '.' || (str[i] == '-' && i > 0))
            return (false);
       if (str[i] == '.')
            hasDecimal = true;
   }
   return (hasDecimal);
}

char* subString(char* str, int left, int right) {
   int i;
   char* subStr = (char*)malloc( sizeof(char) * (right - left +
2));
   for (i = left; i <= right; i++)
      subStr[i - left] = str[i];
   subStr[right - left + 1] = '\0';
   return (subStr);
}

void detectTokens(char* str) {
   int left = 0, right = 0;
   int length = strlen(str);
   while (right <= length && left <= right) {
      if (isDelimiter(str[right]) == false)
      right++;
      if (isDelimiter(str[right]) == true && left == right) {
         if (isOperator(str[right]) == true)
         printf("Valid operator : '%c'\n", str[right]);
         right++;
         left = right;
      } else if (isDelimiter(str[right]) == true && left !=
right || (right == length && left != right)) {
         char* subStr = subString(str, left, right - 1);
         if (isKeyword(subStr) == true)
            printf("keyword : '%s'\n", subStr);
         else if (isInteger(subStr) == true)
            printf("Integer : '%s'\n", subStr);
         else if (isInteger(subStr) == true)
            printf("Number : '%s'\n", subStr);
         else if (isIdentifier(subStr) == true
            && isDelimiter(str[right - 1]) == false)
         printf("Identifier : '%s'\n", subStr);
         else if (isIdentifier(subStr) == false
            && isDelimiter(str[right - 1]) == false)
         printf("Invalid Identifier : '%s'\n", subStr);
         left = right;
      }
   }
   return;
}

int main(){

   FILE* in = fopen("input.txt", "r");
//              ^ open file for reading

    if(!in)
    {
        printf("No such File!");
    }
    else
    {
        printf("All Tokens are : \n");
        char buffer[128];
        size_t num;
        while((num = fread(buffer, sizeof(buffer),
sizeof(*buffer), in)) > 0)
```

```
        {
        // use buffer up to buffer[num]
            detectTokens(buffer);
        }
        // if you are interested if an error occurred or just
the end of file has been read, you can now check e.g. via
        if(!feof(in))
        {
            // error handling
        }
        fclose(in);
    }
   return (0);
}
```

input.txt file:

```
#includde<stdio.h>
#include<conio.h>
void main()
{
  int a,b,ans;
  printf("Enter 1ST No:");
  scanf("%d",&a);
  printf("Enter 2ND No:");
  scanf("%d",&b);
  ans=a+b;
  printf("ANS IS : %d",ans);
  getch();
}
```

Output:



```
All Tokens are :
Identifier : '#includde'
Valid operator : '<'
Identifier : 'stdio.h'
Valid operator : '>'
Identifier : '
#include'
Valid operator : '<'
Identifier : 'conio.h'
Valid operator : '>'
Identifier : '
void'
Identifier : 'main'
Identifier : '
'
Identifier : '
'
keyword : 'int'
Identifier : 'a'
Identifier : 'b'
Identifier : 'ans'
Identifier : '
'
Identifier : 'printf'
Identifier : '"Enter'
Invalid Identifier : '1ST'
Identifier : 'No:"'
Identifier : '
'
Identifier : 'scanf'
Identifier : '"%d"'
Identifier : '&a'
Identifier : '
'
Identifier : 'printf'
Identifier : '"Ent`f    w�
```

4. Write a program to recognize keywords/identifiers in Java (Input File: Java File)

5. Write a program to find First and Follow of the given grammar

```
E --> TE'
E' --> +TE' | e
T --> FT'
T' --> *FT' | e
F --> id | (E)

**e denotes epsilon
```

6. Write a program to construct Predictive parsing table for following

	FIRST	**FOLLOW**
E –> TE’	{ id, (}	{ $,) }
E’ –> +TE’/e	{ +, e }	{ $,) }
T –> FT’	{ id, (}	{ +, $,) }
T’ –> *FT’/e	{ *, e }	{ +, $,) }
F –> id/(E)	{ id, (}	{ *, +, $,) }

7. Write the program to parse the input string ***id+id*id*** using following parsing table

	ID	**+**	*****	**(**	**)**	**$**
E	E–>TE’			E–>TE’		
E’		E’–> +TE’			E’–>e	E’–>e
T	T–>FT’			T–>FT’		
T’		T’–>e	T’–>*FT		T’ –> e	T’ –> e
F	F –> id			F –> (E)		

Where e is epsilon

8. Write a program Intermediate Code Optimization for the expression
   a = b * – c + b * – c.

9. Write a program Intermediate Code Optimization for the expression
   a := (-c * b) + (-c * d)

10. Write a program to generate code for a given
    + a b t1

```
* c d t2
- t1 t2 t
= t ? x
```

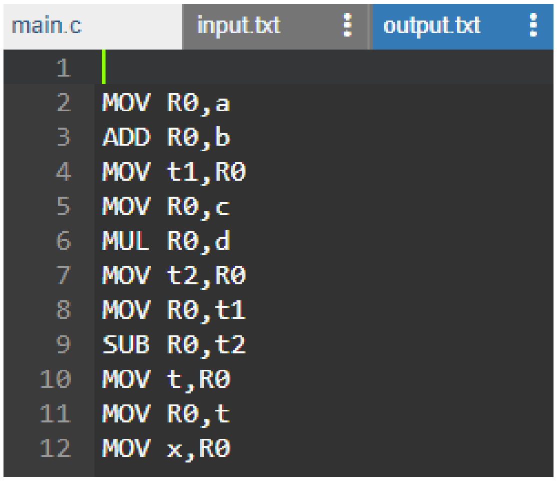
Program:

```
#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);
      fprintf(fp2,"\nMOV %s,R0",result);
    }
     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 R0,%s",arg1);
      fprintf(fp2,"\nMOV %s,R0",result);
    }
    }
    fclose(fp1);
    fclose(fp2);
    getch();
  }
```

input.txt:

op arg1 arg2 result
+ a b t1
* c d t2
- t1 t2 t
= t ? x

Output:



```

MOV R0,a
ADD R0,b
MOV t1,R0
MOV R0,c
MUL R0,d
MOV t2,R0
MOV R0,t1
SUB R0,t2
MOV t,R0
MOV R0,t
MOV x,R0
```

11. Write a program to generate code for a given

+ b c t1
+ t1 e t2
= t ? d

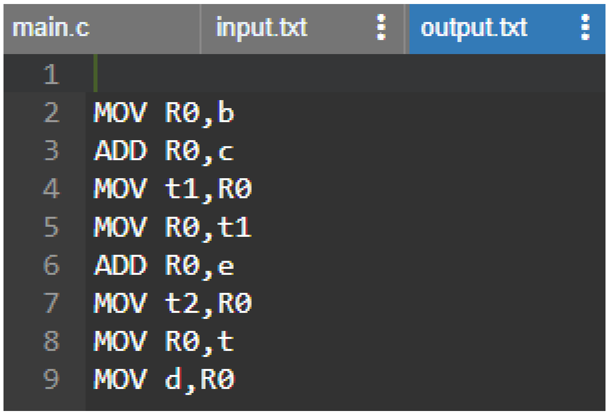
Program:

```
#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);
      fprintf(fp2,"\nMOV %s,R0",result);
    }
     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 R0,%s",arg1);
      fprintf(fp2,"\nMOV %s,R0",result);
    }
    }
    fclose(fp1);
    fclose(fp2);
    getch();
  }
```

input.txt:

op arg1 arg2 result
+ b c t1
+ t1 e t2
= t ? d

Output:



```

MOV R0,b
ADD R0,c
MOV t1,R0
MOV R0,t1
ADD R0,e
MOV t2,R0
MOV R0,t
MOV d,R0
```

12. Generate intermediate code for following code for pass1 of assembler.

Input.txt

```
COPY START        1000
-       LDA   ALPHA
-       ADD   ONE
-       SUB   TWO
-       STA   BETA
ALPHA       BYTE  C'KLNCE
ONE   RESB  2
TWO   WORD       5
BETA  RESW 1
-       END  -
```

Program:

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.HashMap;
import java.io.BufferedReader;
import java.io.BufferedWriter;

public class Main {
    int lc=0;
    int libtab_ptr=0,pooltab_ptr=0;
    int symIndex=0,litIndex=0;
    LinkedHashMap<String, TableRow> SYMTAB;
    ArrayList<TableRow> LITTAB;
    ArrayList<Integer> POOLTAB;
    private BufferedReader br;

    public Main()
    {
        SYMTAB =new LinkedHashMap<>();
        LITTAB=new ArrayList<>();
        POOLTAB=new ArrayList<>();
        lc=0;
        POOLTAB.add(0);
    }
    public static void main(String[] args) {
        Main one=new Main();
        try {
            one.parseFile();
        }
        catch (Exception e) {
            System.out.println("Error: "+e);
        }
    }
    public class INSTtable {
        HashMap<String, Integer> AD,RG,IS,CC,DL;
        public INSTtable()
        {
            AD=new HashMap<>();
            CC = new HashMap<>();
            IS = new HashMap<>();
            RG = new HashMap<>();
            DL=new HashMap<String, Integer>();
            DL.put("DC", 01);
            DL.put("DS", 02);
            IS.put("STOP",0);
            IS.put("ADD",1);
            IS.put("SUB",2);
            IS.put("MULT",3);
            IS.put("MOVER",4);
            IS.put("MOVEM",5);
            IS.put("COMP",6);
            IS.put("BC",7);
            IS.put("DIV",8);
            IS.put("READ",9);
            IS.put("PRINT",10);
            CC.put("LT",1);
            CC.put("LE",2);
            CC.put("EQ",3);
            CC.put("GT",4);
            CC.put("GE",5);
            CC.put("ANY",6);
            AD.put("START",1);
            AD.put("END",2);
            AD.put("ORIGIN",3);
            AD.put("EQU",4);
            AD.put("LTORG",5);
            RG.put("AREG",1);
            RG.put("BREG",2);
            RG.put("CREG",3);
            RG.put("DREG",4);
        }
        public String getType(String s) {
            s=s.toUpperCase();
            if(AD.containsKey(s))
            return "AD";
            else if(IS.containsKey(s))
            return "IS";
            else if(CC.containsKey(s))
            return "CC";
            else if(DL.containsKey(s))
            return "DL";
            else if(RG.containsKey(s))
            return "RG";
            return "";
        }

        public int getCode(String s) {
            s = s.toUpperCase();
            if(AD.containsKey(s))
            return AD.get(s);
            else if(IS.containsKey(s))
            return IS.get(s);
            else if(CC.containsKey(s))
            return CC.get(s);
            else if(DL.containsKey(s))
            return DL.get(s);
            else if(RG.containsKey(s))
            return RG.get(s);
            return -1;
        }
    }
    public class TableRow {
        String symbol;
        int addess,index;
        public String getSymbol() {
            return symbol;
        }
        public TableRow(String symbol, int addess) {
            super();
            this.symbol = symbol;
            this.addess = addess;
            index=0;
        }
        public void setSymbol(String symbol) {
            this.symbol = symbol;
        }
```

```
        public TableRow(String symbol, int addess, int index) {
            super();
            this.symbol = symbol;
            this.addess = addess;
            this.index = index;
        }
        public int getAddess() {
            return addess;
        }
        public void setAddess(int addess) {
            this.addess = addess;
        }
        public int getIndex() {
            return index;
        }
        public void setIndex(int index) {
            this.index = index;
        }
    }
    public void parseFile() throws Exception {
        String prev="";
        String line,code;
        br = new BufferedReader(new FileReader("input.asm"));
        BufferedWriter bw=new BufferedWriter(new
FileWriter("IC.txt"));
        INSTtable lookup=new INSTtable();

        while((line=br.readLine())!=null) {
            String parts[]=line.split("\\s+");
            if(!parts[0].isEmpty())
            {
                if(SYMTAB.containsKey(parts[0]))
                SYMTAB.put(parts[0], new TableRow(parts[0], lc,
SYMTAB.get(parts[0]).getIndex()));
                else
                SYMTAB.put(parts[0],new TableRow(parts[0], lc,
++symIndex));
            }

            if(parts[1].equals("LTORG"))
            {
                int ptr=POOLTAB.get(pooltab_ptr);
                for(int j=ptr;j<libtab_ptr;j++)
                {
                    lc++;
                    LITTAB.set(j, new
TableRow(LITTAB.get(j).getSymbol(),lc));

code="(DL,01)\t(C,"+LITTAB.get(j).symbol+")";
                    bw.write(code+"\n");
                }
                pooltab_ptr++;
                POOLTAB.add(libtab_ptr);
            }
            if(parts[1].equals("START"))
            {
                lc=expr(parts[2]);
                code="(AD,01)\t(C,"+lc+")";
                bw.write(code+"\n");
                prev="START";
            }
            else if(parts[1].equals("ORIGIN"))
            {
                lc=expr(parts[2]);
                String splits[]=parts[2].split("\\+");

code="(AD,03)\t(S,"+SYMTAB.get(splits[0]).getIndex()+")+"+Integ
er.parseInt(splits[1]);
                bw.write(code+"\n");
            }

            if(parts[1].equals("EQU"))
            {
                int loc=expr(parts[2]);
                if(parts[2].contains("+"))
                {
                    String splits[]=parts[2].split("\\+");

code="(AD,04)\t(S,"+SYMTAB.get(splits[0]).getIndex()+")+"+Integ
er.parseInt(splits[1]);
                }
                else if(parts[2].contains("-"))
                {
                    String splits[]=parts[2].split("\\-");

code="(AD,04)\t(S,"+SYMTAB.get(splits[0]).getIndex()+")-"+Integ
er.parseInt(splits[1]);
                }
                else
                {
```

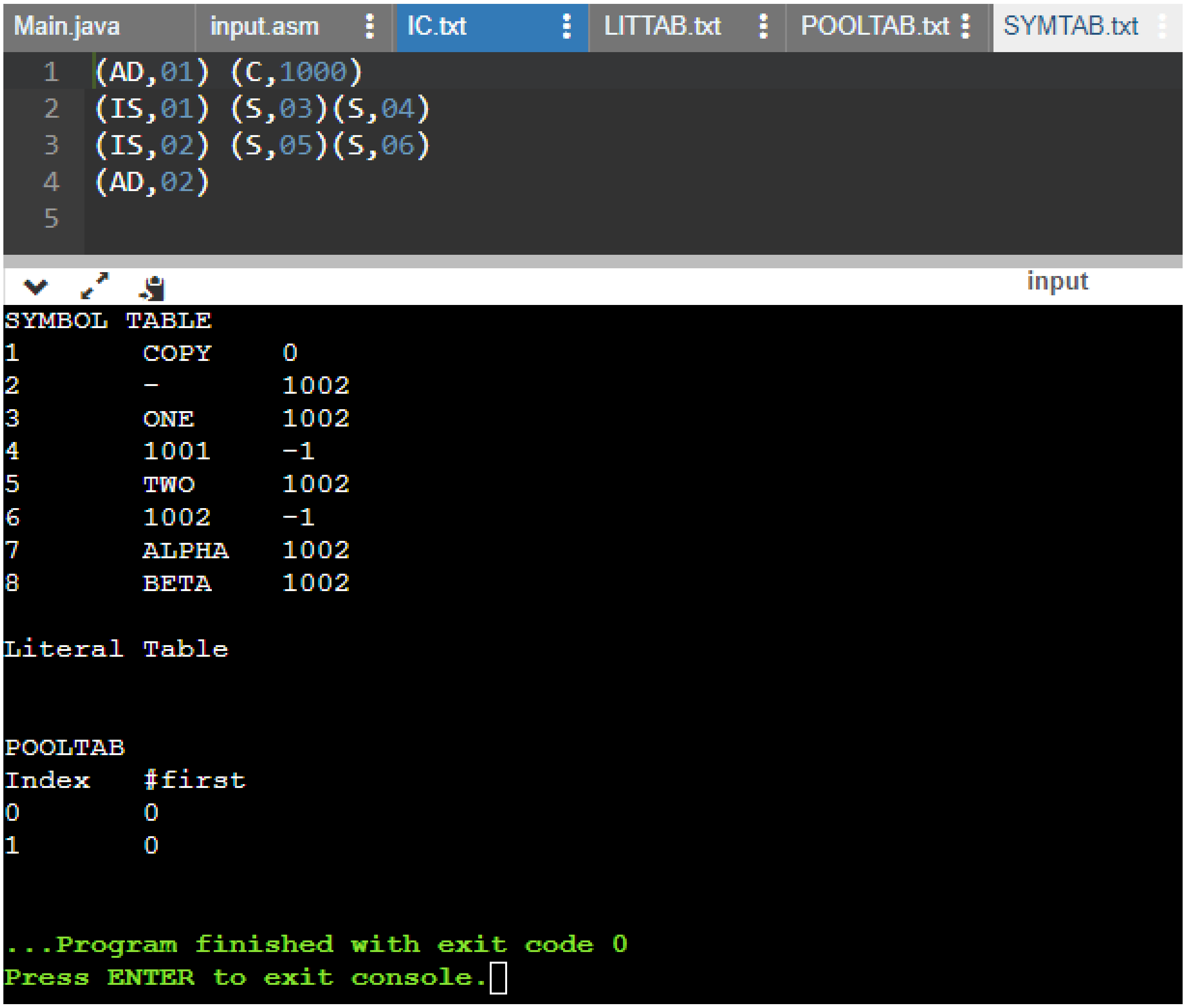
```
code="(AD,04)\t(C,"+Integer.parseInt(parts[2]+")");
                }
                bw.write(code+"\n");
                if(SYMTAB.containsKey(parts[0]))
                    SYMTAB.put(parts[0], new 
TableRow(parts[0],loc,SYMTAB.get(parts[0]).getIndex())) ;
                else
                    SYMTAB.put(parts[0], new 
TableRow(parts[0],loc,++symIndex));
            }

            if(parts[1].equals("DC"))
            {
                lc++;
                int 
constant=Integer.parseInt(parts[2].replace("'",""));
                code="(DL,01)\t(C,"+constant+")";
                bw.write(code+"\n");
            }
            else if(parts[1].equals("DS"))
            {
                int size=Integer.parseInt(parts[2].replace("'", 
""));
                code="(DL,02)\t(C,"+size+")";
                bw.write(code+"\n");
                lc=lc+size;
                prev="";
            }
            if(lookup.getType(parts[1]).equals("IS"))
            {
                code="(IS,0"+lookup.getCode(parts[1])+")\t";
                int j=2;
                String code2="";
                while(j<parts.length)
                {
                    parts[j]=parts[j].replace(",", "");
                    if(lookup.getType(parts[j]).equals("RG"))
                    {
                        code2+=lookup.getCode(parts[j])+"\t";
                    }
                    else
                    {
                        if(parts[j].contains("="))
                        {
                            parts[j]=parts[j].replace("=",
"").replace("'", "");
                            LITTAB.add(new TableRow(parts[j],
-1,++litIndex));
                            libtab_ptr++;
                            code2+="(L,"+(litIndex)+")";
                        }
                        else if(SYMTAB.containsKey(parts[j]))
                        {
                            int
ind=SYMTAB.get(parts[j]).getIndex();
                            code2+= "(S,0"+ind+")";
                        }
                        else
                        {
                            SYMTAB.put(parts[j], new
TableRow(parts[j],-1,++symIndex));
                            int
ind=SYMTAB.get(parts[j]).getIndex();
                            code2+= "(S,0"+ind+")";
                        }
                    }
                    j++;
                }
                lc++;
                code=code+code2;
                bw.write(code+"\n");
            }

            if(parts[1].equals("END"))
            {
                int ptr=POOLTAB.get(pooltab_ptr);
                for(int j=ptr;j<libtab_ptr;j++)
                {
                    lc++;
                    LITTAB.set(j, new
TableRow(LITTAB.get(j).getSymbol(),lc));

code="(DL,01)\t(C,"+LITTAB.get(j).symbol+")";
                    bw.write(code+"\n");
                }
                pooltab_ptr++;
                POOLTAB.add(libtab_ptr);
                code="(AD,02)";
                bw.write(code+"\n");
            }
        }
        bw.close();
        printSYMTAB();

        PrintLITTAB();
        printPOOLTAB();
    }
    void PrintLITTAB() throws IOException
    {
        BufferedWriter bw=new BufferedWriter(new
FileWriter("LITTAB.txt"));
        System.out.println("\nLiteral Table\n");

        for(int i=0;i<LITTAB.size();i++)
        {
            TableRow row=LITTAB.get(i);

System.out.println(i+"\t"+row.getSymbol()+"\t"+row.getAddess())
;

bw.write((i+1)+"\t"+row.getSymbol()+"\t"+row.getAddess()+"\n");
        }
        bw.close();
    }
    void printPOOLTAB() throws IOException
    {
        BufferedWriter bw=new BufferedWriter(new
FileWriter("POOLTAB.txt"));
        System.out.println("\nPOOLTAB");
        System.out.println("Index\t#first");
        for (int i = 0; i < POOLTAB.size(); i++) {
            System.out.println(i+"\t"+POOLTAB.get(i));
            bw.write((i+1)+"\t"+POOLTAB.get(i)+"\n");
        }
        bw.close();
    }
    void printSYMTAB() throws IOException
    {
        BufferedWriter bw=new BufferedWriter(new
FileWriter("SYMTAB.txt"));
        //Printing Symbol Table
        java.util.Iterator<String> iterator =
SYMTAB.keySet().iterator();
        System.out.println("SYMBOL TABLE");
        while (iterator.hasNext()) {
            String key = iterator.next().toString();
            TableRow value = SYMTAB.get(key);
            System.out.println(value.getIndex()+"\t" +
value.getSymbol()+"\t"+value.getAddess());
            bw.write(value.getIndex()+"\t" +
value.getSymbol()+"\t"+value.getAddess()+"\n");
        }
        bw.close();
    }

    public int expr(String str)
    {
        int temp=0;
        if(str.contains("+"))
        {
            String splits[]=str.split("\\+");

temp=SYMTAB.get(splits[0]).getAddess()+Integer.parseInt(splits[
1]);
        }
        else if(str.contains("-"))
        {
            String splits[]=str.split("\\-");

temp=SYMTAB.get(splits[0]).getAddess()-(Integer.parseInt(splits
[1]));
        }
        else
        {
            temp=Integer.parseInt(str);
        }
        return temp;
    }
}
```

input.txt:

```
COPY  START       1000
-   LDA     ALPHA  1000
-   ADD  ONE    1001
-   SUB     TWO    1002
-   STA     BETA   1003
ALPHA BYTE  C'KLNCE 1004
```

```
ONE     RESB  2      1006
TWO   WORD  5       1011
BETA  RESW  1      1012
-    END        -      1013
```

Output:



```
(AD,01) (C,1000)
(IS,01) (S,03)(S,04)
(IS,02) (S,05)(S,06)
(AD,02)
```

```
SYMBOL TABLE
1       COPY    0
2       -       1002
3       ONE     1002
4       1001    -1
5       TWO     1002
6       1002    -1
7       ALPHA   1002
8       BETA    1002

Literal Table


POOLTAB
Index   #first
0       0
1       0


...Program finished with exit code 0
Press ENTER to exit console.
```

13. Write a program to generate MOT, POT, ST and LT for following code(Pass1)

Input.txt

```
COPY START      1000
-       LDA   ALPHA
-       ADD   ONE
-       SUB   TWO
-       STA   BETA
ALPHA       BYTE  C'KLNCE
ONE   RESB  2
TWO   WORD       5
BETA  RESW 1
-       END   -
```

14. Write a program to generate MOT, POT, ST and LT for following code (Pass1)

Input.txt

```
PG1          START       1000
             USING       *, 15
             L            1, FOUR
             A            1, ='5'
             ST           1, TEMP
             A            1, ='4'
FOUR        DC            F'4'
TEMP        DS            '1'F
             END
```

15. Write a program for the implementation of pass two of a two pass assembler

input.txt:			
-	COPY	START	1000
1000	-	LDA	ALPHA
1003	-	ADD	ONE
1006	-	SUB	TWO
1009	-	STA	BETA
1012	ALPHA	BYTE	C'KLNCE
1017	ONE	RESB	2
1019	TWO	WORD	5
1022	BETA	RESW	1
1025	-	END	-

symbol.txt:	
1012	ALPHA
1017	ONE
1019	TWO
1022	BETA

optab.txt:	

LDA	00
STA	23
ADD	01
SUB	05

16. Write a program for the implementation of pass two of a two pass assembler

    INPUT FILES

    INTERMEDIATE.DAT

```
START 2000
2000  LDA FIVE
2003  STA ALPHA
2006  LDCH CHARZ
2009  STCH C1
2012 ALPHA RESW 1
2015 FIVE WORD 5
2018 CHARZ BYTE C'EOF'
2019 C1 RESB 1
2020  END
```

OPTAB.DAT

```
LDA 33
STA 44
LDCH 53
STCH 57
END
```

SYMTAB.DAT

```
ALPHA 2012
FIVE 2015
CHARZ 2018
C1 2019
```

17. Write a program for the implementation of pass two of a two pass assembler

INPUT FILES:

INTERMED.DAT

```
COPY START    2000
2000     LDA         FIVE
2003     STA         ALPHA
2006     LDCH        CHARZ
2009     STCH        C1
2012   ALPHA    RESW     1
2015   FIVE    WORD      5
2018   CHARZ    BYTE     C'EOF'
2019   C1    RESB   1
2020     END
```

SYMTAB.DAT

```
ALPHA       2012
FIVE  2015
CHARZ       2018
C1      2019
```

18. Write a program to define MDT and output of Macro processor

```
CALC START 1000
SUM MACRO
 LDA #5
 ADD #10
 STA 2000
 MEND
 LDA LENGTH
 COMP ZERO
 JEQ LOOP
 SUM
LENGTH WORD S
ZERO WORD S
LOOP SUM
 END
```

19. Write a program to define MDT, and output of Macro processor

```
MACRO ADD1
MOV A,B
ADD C
MEND
MACRO SUB1
STORE C
MEND
MOV B, 10
MOV C, 20
ADD1
MUL C
SUB1
END
```

20. Write a program to define MDT, MNT and ALA Data structure of Macro processor

```
MACRO ADD1
MOV A,B
ADD C
MEND
MACRO SUB1
STORE C
MEND
MOV B, 10
MOV C, 20
ADD1
MUL C
SUB1
END
```