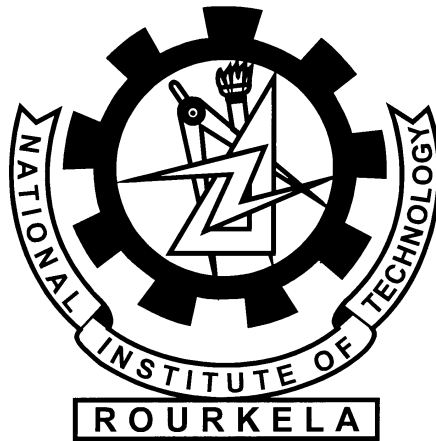


**Compiler Design Laboratory**  
**6<sup>th</sup> Semester 2018**

Submitted by-  
Dattatreya Tripathy  
Roll number: 115CS0250



Department of Computer Science and Engineering  
National Institute of Technology, Rourkela

**Date: 02-01-2018**

1. Design a scanner using flex to count the number of ASCII characters, words, lines, punctuations and “\n” in a given string –

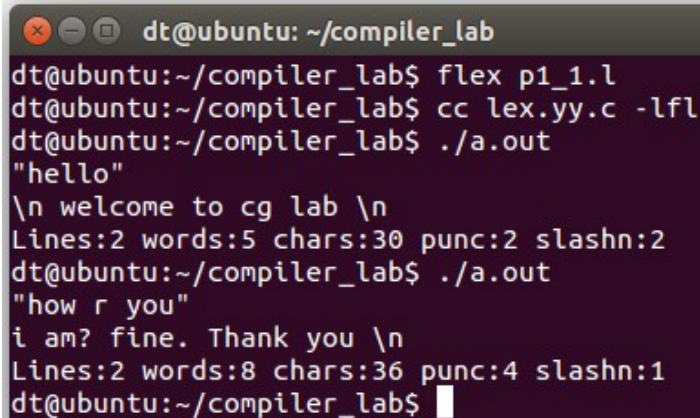
Symbol set for words: [a-zA-Z0-9]+

Punctuation symbols: .,:;?'''

\n newline character

**Program:**

```
p1_1.l
/* Counting number of lines, words, punctuations etc... */
%{
int chars = 0;
int words = 0;
int lines = 0;
int punc = 0;
int slashn = 0;
}%
%%
[a-zA-Z0-9]+ { words++; chars += strlen(yytext); }
\n          { chars++; lines++; }
['":;?!.] + { chars++; punc++; }
\\n         { chars++; slashn++; }
.           { chars++; }
%%
main(int argc, char **argv)
{
yylex();
printf("Lines:%d words:%d chars:%d punc:%d slashn:%d \n", lines,
words, chars, punc, slashn);
}
```



A terminal window titled 'dt@ubuntu: ~/compiler\_lab' showing the execution of a flex scanner program. The user runs 'flex p1\_1.l', then 'cc lex.yy.c -lfl', and finally './a.out'. The program outputs the counts for the input strings 'hello' and 'how r you'.

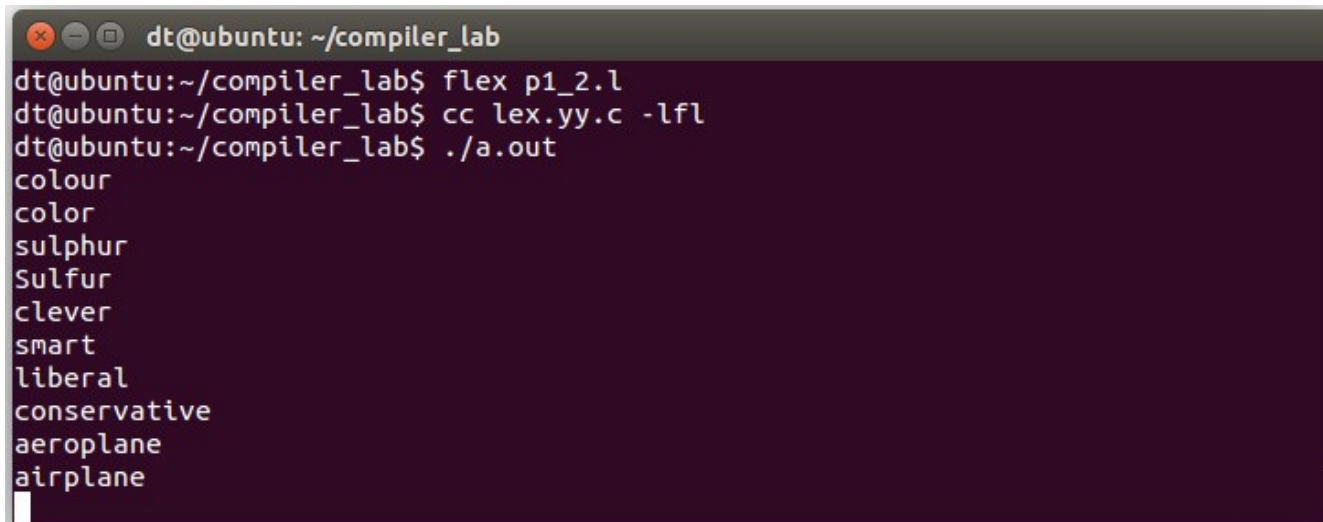
```
dt@ubuntu:~/compiler_lab$ flex p1_1.l
dt@ubuntu:~/compiler_lab$ cc lex.yy.c -lfl
dt@ubuntu:~/compiler_lab$ ./a.out
"hello"
\n welcome to cg lab \n
Lines:2 words:5 chars:30 punc:2 slashn:2
dt@ubuntu:~/compiler_lab$ ./a.out
"how r you"
i am? fine. Thank you \n
Lines:2 words:8 chars:36 punc:4 slashn:1
dt@ubuntu:~/compiler_lab$
```

2. Design a pattern matcher. The pattern matcher should take the British English version of a word and provide its corresponding American English version.

**Program:**

```
p1_2.l
/* English -> American */

%%
"colour"      { printf("color"); }
"flavour"     { printf("flavor"); }
"clever"      { printf("smart"); }
"smart"       { printf("elegant"); }
"liberal"     { printf("conservative"); }
"aeroplane"   { printf("airplane"); }
"biscuit"     { printf("cookie"); }
"car park"    { printf("parking lot"); }
"chips"       { printf("French fries"); }
"sulphur"     { printf("Sulfur"); }
.             { printf("%s", yytext); }
%%
```



```
dt@ubuntu: ~/compiler_lab
dt@ubuntu:~/compiler_lab$ flex p1_2.l
dt@ubuntu:~/compiler_lab$ cc lex.yy.c -lfl
dt@ubuntu:~/compiler_lab$ ./a.out
colour
color
sulphur
Sulfur
clever
smart
liberal
conservative
aeroplane
airplane
```

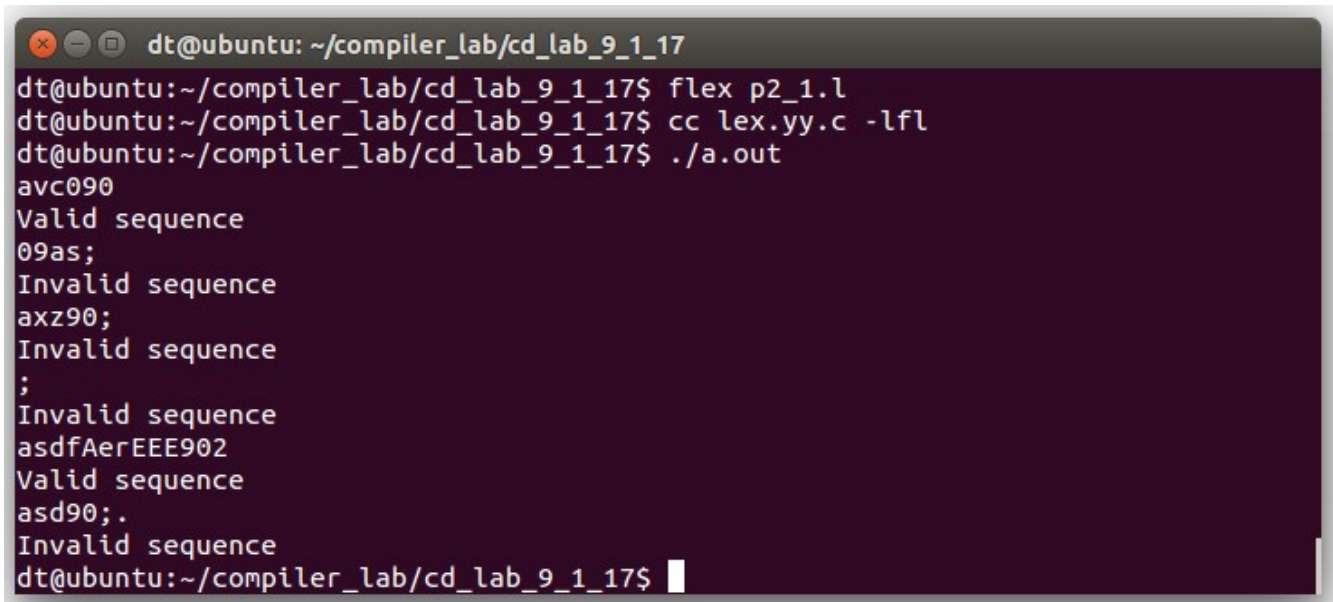
**Date: 09-01-2018**

1. Write a FLEX application that will recognize valid identifiers/sequences having symbols a-z, A-Z, 0-9 and underscore (\_). Each identifier should not start with a symbol from 0-9. Any other symbol like decimal point/ full stop (.), comma (,), semicolon (;), colon (:), +, -, \*, /, @, \$, & are considered invalid .

**Program:**

```
p2_1.l
/*FLex application to recognize valid and invalid sequence based on input*/
%{

%}
%%
[a-zA-Z_]+[a-zA-Z0-9_]*+ {printf("Valid sequence\n");}
[\n ] { }
[a-zA-Z_]([a-zA-Z0-9_.,;+*/@$&-])*+ {printf("Invalid sequence\n");}
[0-9.,;:(+/-/@$]([a-zA-Z]|[0-9]|_|[.,;:(+/-/@$])* {printf("Invalid
sequence\n");}
%%
main(int argc, char **argv)
{
    yylex();
}
```



```
dt@ubuntu: ~/compiler_lab/cd_lab_9_1_17
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$ flex p2_1.l
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$ cc lex.yy.c -lfl
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$ ./a.out
avc090
Valid sequence
09as;
Invalid sequence
axz90;
Invalid sequence
;
Invalid sequence
asdfAerEEE902
Valid sequence
asd90;.
Invalid sequence
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$
```

2. Write a FLEX application that will recognize strings of odd numbers of 0's and even numbers of 1's.

**Program:**

```
p2_2.1
/*Flex application to check valid or invalid sequence based on number of
zeros and ones*/
%{
    int zeros = 0;
    int ones = 0;
}%
%%
[0] {zeros++;}
[1] {ones++;}
[ \n ] { }
. {printf("Invalid character\n");}
%%
main(int argc, char **argv)
{
    yylex();
    printf("Zeros=%d and ones=%d\n", zeros, ones);
    if(zeros%2==1 && ones%2==0)
        printf("Valid sequence with odd number of zeros and even
number of ones\n");
    else
        printf("Invalid sequence without odd number of zeros and even
number of ones\n");
}
```

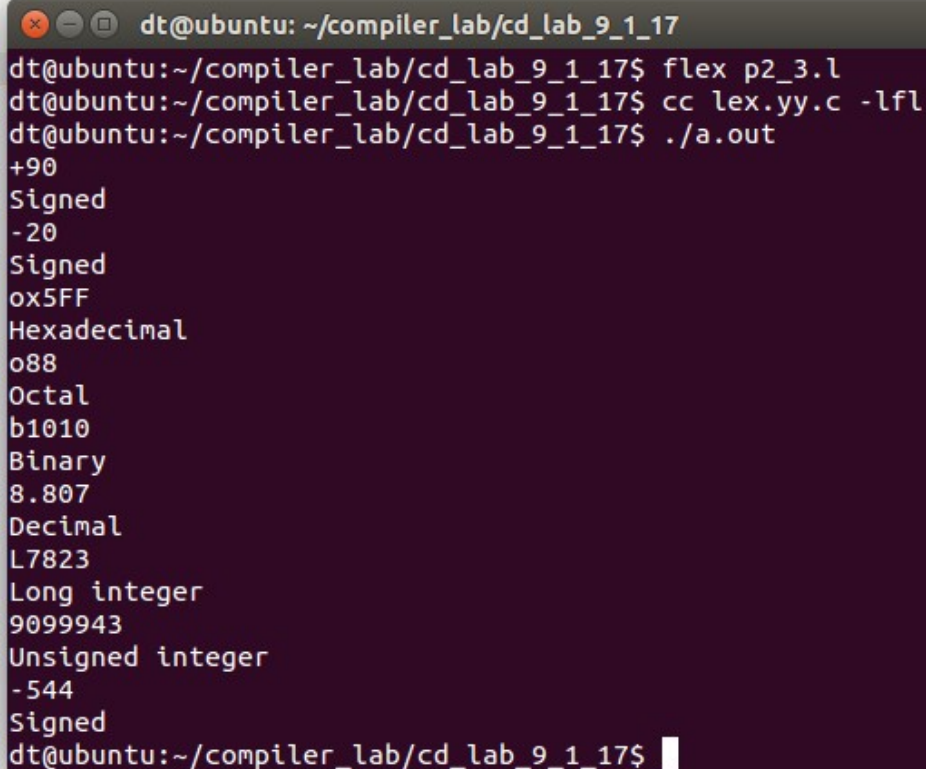
```
dt@ubuntu: ~/compiler_lab/cd_lab_9_1_17
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$ ./a.out
01001
Zeros=3 and ones=2
Valid sequence with odd number of zeros and even number of ones
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$ ./a.out
0000110
Zeros=5 and ones=2
Valid sequence with odd number of zeros and even number of ones
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$ ./a.out
00111
Zeros=2 and ones=3
Invalid sequence without odd number of zeros and even number of ones
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$ ./a.out
0011
Zeros=2 and ones=2
Invalid sequence without odd number of zeros and even number of ones
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$
```

3. Write a FLEX application that will identify signed/ unsigned integer and long integer constants in decimal, hexadecimal, binary and octal representations used in C language.

**Program:**

```
p2_3.1
/*Flex application to identify signed, unsigned, hex, oct etc as in c*/
%{

%}
Sign [+|-][0-9]([0-9])*+
Hexadecimal [0][x|X][0-9A-F]([0-9A-F])*+
Octal [0][0-9]([0-9])*+
Binary [b|B][0-1]([0-1])*+
Decimal [0-9]([0-9])*[.][0-9]([0-9])*+
Long [l|L][0-9]([0-9])*+
Unsigned [0-9]([0-9])*+
%%
{Sign} printf("Signed\n");
{Hexadecimal} printf("Hexadecimal\n");
{Octal} printf("Octal\n");
{Binary} printf("Binary\n");
{Decimal} printf("Decimal\n");
{Long} printf("Long integer\n");
{Unsigned} printf("Unsigned integer\n");
[ \n ] { }
. printf("Invalid\n");
%%
main(int argc, char **argv)
{
    yylex();
}
```



```
dt@ubuntu: ~/compiler_lab/cd_lab_9_1_17
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$ flex p2_3.1
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$ cc lex.yy.c -lfl
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$ ./a.out
+90
Signed
-20
Signed
0x5FF
Hexadecimal
088
Octal
b1010
Binary
8.807
Decimal
L7823
Long integer
9099943
Unsigned integer
-544
Signed
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$
```

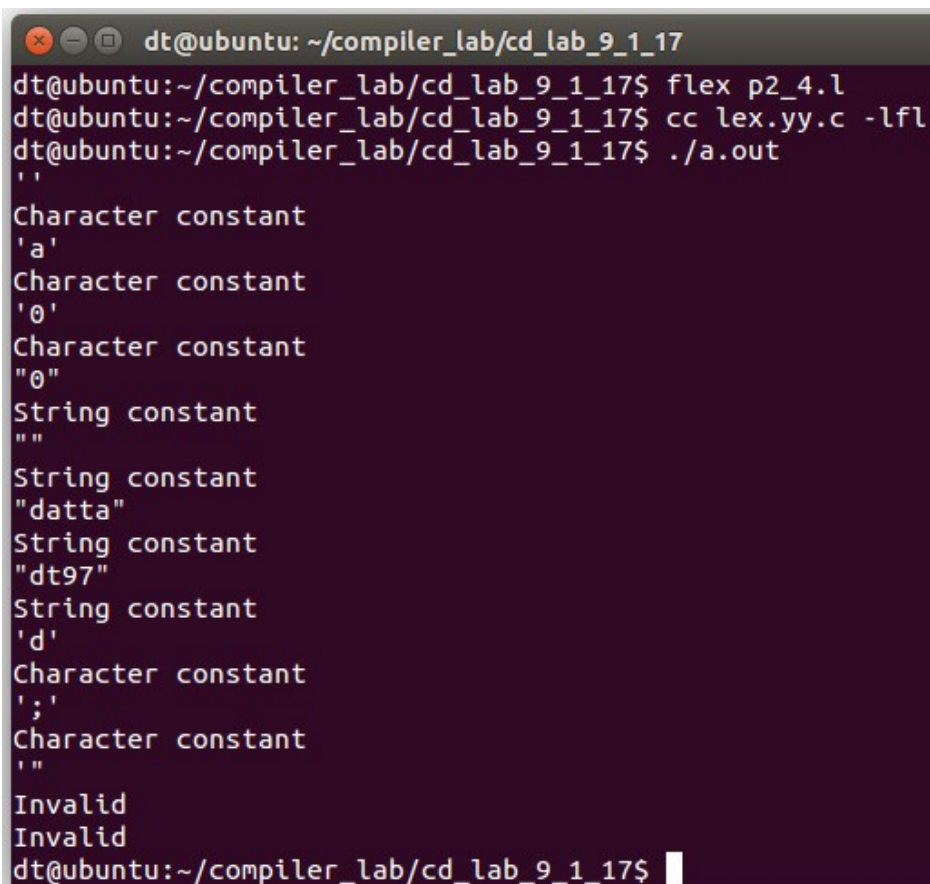
4. Write a FLEX program that will identify character and string constants as defined in C language.

**Program:**

```
p2_4.1
/*Flex program to identify character and string constant as in c*/
%{

%}
Character '['[a-zA-Z0-9;,.:]['']+
String '['"[a-zA-Z0-9;,.:]*["]+
%%

\n { }
{Character}|[']['] printf("Character constant\n");
{String} printf("String constant\n");
. printf("Invalid\n");
%%
main(int argc, char **argv)
{
    yylex();
}
```



```
dt@ubuntu: ~/compiler_lab/cd_lab_9_1_17
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$ flex p2_4.1
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$ cc lex.yy.c -lfl
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$ ./a.out
'
Character constant
'a'
Character constant
'0'
Character constant
"0"
String constant
""
String constant
"datta"
String constant
"dt97"
String constant
'd'
Character constant
';'
Character constant
'"
Invalid
Invalid
dt@ubuntu:~/compiler_lab/cd_lab_9_1_17$
```

**Date: 16-01-2018**

1. Design an application using FLEX and BISON that will do the following
  - (i) check the correctness of a Boolean expression,
  - (ii) express it in standard Sum of Product form, and
  - (iii) convert the Sum of Product representation to standard Product of Sum form.

Example1

Input:

$F(x,y) = x'.y + x.y'$

Output:

The given expression is correct.

Sum of Product:  $x'.y + x.y'$

Product of sum:  $(x+y).(x'+y')$

Example2

Input:

$F(x,y) = x.y +$

Output:

Error 1: Incomplete expression, operand missing at x.y +

Symbols:

[a-z] as boolean variables.

operators

' complement

+ or

. and

other symbols

( left parenthesis

) right parenthesis

F(...) function with list of variables

= assignment

**Program:**

p3\_1.1

/\*Check validity of boolean expression\*/

%{

#include "p3\_1.tab.h"

%}

BVAR [a-z]

%%

{BVAR} { return BOOLVAR; }

"" { return NOT; }

"+" { return OR; }

"." { return AND; }

"(" { return LPAREN; }

")" { return RPAREN; }



```

"=" { return ASSIGN; }
"F" { return F; }
"," { return COMMA; }
\n { return EOL; }
[ \t] { }
. { printf("Unknown Character/String\n"); }
%%

```

```

p3_1.y
%{
#include <stdio.h>
%}
%token BOOLVAR
%token NOT
%token OR
%token AND
%token LPAREN
%token RPAREN
%token ASSIGN
%token F
%token COMMA
%token EOL
%%

```

```

beg:
| beg F LPAREN s RPAREN ASSIGN exp EOL { printf("Valid boolean
expression\n"); }
;
s: BOOLVAR
| s COMMA BOOLVAR
;
exp: term
| exp OR term
;
term: factor
| term AND factor
;
factor: BOOLVAR
;
factor: factor NOT
;
factor: LPAREN exp RPAREN

```

```

;
%%
main(int argc, char **argv)
{
    yyparse();
}
yyerror(char *s1)
{
    fprintf(stderr,"error: %s\n",s1);
}
yywrap()
{
    return 1;
}

```

```

p3_1.sh
bison -d p3_1.y
flex p3_1.l
cc -o p3_1test p3_1.tab.c lex.yy.c -lfl

```

```

dt@ubuntu: ~/compiler_lab
dt@ubuntu:~/compiler_lab$ ./p3_1.sh
dt@ubuntu:~/compiler_lab$ ./p3_1test
F(x, y) = x'.y+x.y'
Valid boolean expression
F(x) = xx
error: syntax error
dt@ubuntu:~/compiler_lab$ ./p3_1test
F(x, y, z) = x'+y' + z'
Valid boolean expression
F(x, z) = x'z+y(x+z.z'+x'.y + z')
error: syntax error
dt@ubuntu:~/compiler_lab$ ./p3_1test
F(x, y, z) = x'.z+y.(x+z.z'+x'.y + z')
Valid boolean expression
dt@ubuntu:~/compiler_lab$

```

2. Design a machine that will recognize numbers between 0-999. For example: "Two hundred thirty nine" is a correct form of 239 in words. It should also detect an incorrect input "Twelve hundred twenty two."

**Program:**

p3\_2.1

```
/* Flex and bison application to recognise numbers from 1 to 999 in words */
```

```
%{
```

```
#include "p3_2.tab.h"
```

```
%}
```

```
%%
```

```
"hundred" {return(HUNDRED);}
```

```
"one" {return(ONE);}
```

```
"two" {return(TWO);}
```

```
"three" {return(THREE);}
```

```
"four" {return(FOUR);}
```

```
"five" {return(FIVE);}
```

```
"six" {return(SIX);}
```

```
"seven" {return(SEVEN);}
```

```
"eight" {return(EIGHT);}
```

```
"nine" {return(NINE);}
```

```
"ten" {return(TEN);}
```

```
"zero" {return(ZERO);}
```

```
"eleven" {return(ELEVEN);}
```

```
"twelve" {return(TWELVE);}
```

```
"thirteen" {return(THIRTEEN);}
```

```
"fourteen" {return(FOURTEEN);}
```

```
"fifteen" {return(FIFTEEN);}
```

```
"sixteen" {return(SIXTEEN);}
```

```
"seventeen" {return(SEVENTEEN);}
```

```
"eighteen" {return(EIGHTEEN);}
```

```
"nineteen" {return(NINETEEN);}
```

```
"twenty" {return(TWENTY);}
```

```
"thirty" {return(THIRTY);}
```

```
"forty" {return(FORTY);}
```

```
"fifty" {return(FIFTY);}
```

```
"sixty" {return(SIXTY);}
```

```
"seventy" {return(SEVENTY);}
```

```
"eighty" {return(EIGHTY);}
```

```
"ninety" {return(NINETY);}
```

```
\n {return EOL;}
```

```
[ \t] { }
```

```
. {printf("Unknown Character/String");}
```

%%

p3\_2.y

```
%{  
#include<stdio.h>  
%}  
%token HUNDRED  
%token ONE  
%token TWO  
%token THREE  
%token FOUR  
%token FIVE  
%token SIX  
%token SEVEN  
%token EIGHT  
%token NINE  
%token N  
%token ZERO  
%token TEN  
%token ELEVEN  
%token TWELVE  
%token THIRTEEN  
%token FOURTEEN  
%token FIFTEEN  
%token SIXTEEN  
%token SEVENTEEN  
%token EIGHTEEN  
%token NINETEEN  
%token TWENTY  
%token THIRTY  
%token FORTY  
%token FIFTY  
%token SIXTY  
%token SEVENTY  
%token EIGHTY  
%token NINETY  
%token EOL  
%%
```

beg:

```
| beg s EOL {printf("\n");}  
;  
s: u  
;  
u: t d  
| tp  
| dp  
;  
t: h x
```

```

;
tp: h y
;
h:
| dpp
| d HUNDRED {printf("hundred");}
;
d:
| ONE {printf("one");}
| TWO {printf("two");}
| THREE {printf("three");}
| FOUR {printf("four");}
| FIVE {printf("five");}
| SIX {printf("six");}
| SEVEN {printf("seven");}
| EIGHT {printf("eight");}
| NINE {printf("nine");}
;
dp: ONE {printf("one");}
| TWO {printf("two");}
| THREE {printf("three");}
| FOUR {printf("four");}
| FIVE {printf("five");}
| SIX {printf("six");}
| SEVEN {printf("seven");}
| EIGHT {printf("eight");}
| NINE {printf("nine");}
| TEN {printf("ten");}
;
x:
| TWENTY {printf("twenty");}
| THIRTY {printf("thirty");}
| FORTY {printf("forty");}
| FIFTY {printf("fifty");}
| SIXTY {printf("sixty");}
| SEVENTY {printf("seventy");}
| EIGHTY {printf("eighty");}
| NINETY {printf("ninety");}
;
y: TEN {printf("ten");}
| ELEVEN {printf("eleven");}
| TWELVE {printf("twelve");}
| THIRTEEN {printf("thirteen");}
| FOURTEEN {printf("fourteen");}
| FIFTEEN {printf("fifteen");}
| SIXTEEN {printf("sixteen");}
| SEVENTEEN {printf("seventeen");}
| EIGHTEEN {printf("eighteen");}
| NINETEEN {printf("nineteen");}

```

```

;
dpp: ONE {printf("one");}
| TWO {printf("two");}
| THREE {printf("three");}
| FOUR {printf("four");}
| FIVE {printf("five");}
| SIX {printf("six");}
| SEVEN {printf("seven");}
| EIGHT {printf("eight");}
| NINE {printf("nine");}
;
%%
main(int argc, char **argv)
{
    yyparse();
}
yyerror(char *s1)
{
    fprintf(stderr, "error: %s\n", s1);
}
yywrap()
{
    return 1;
}

```

```

p3_2.sh
# part of the makefile
#p3_1calci: p3_1calci.l p3_1calci.y
#p3_1calci.l p3_1calci.y
bison -d p3_2.y
flex p3_2.l
cc -o p3_2test p3_2.tab.c lex.yy.c -lfl

```

```
dt@ubuntu: ~/compiler_lab
dt@ubuntu:~/compiler_lab$ ./p3_2test
one hundred two
onehundredtwo
nine hundred
ninehundred
nine hundred ninety nine
ninehundredninetynine
one thousand
Unknown Character/StringUnknown Character/StringUnknown Character/StringUnknown
Character/StringUnknown Character/StringUnknown Character/StringUnknown Characte
r/StringUnknown Character/Stringone
six hundred forty
sixhundredforty
one
one
nine
nine
eoght
Unknown Character/StringUnknown Character/StringUnknown Character/StringUnknown
Character/StringUnknown Character/String
eight
eight
dt@ubuntu:~/compiler_lab$
```

**Date: 23-01-2016**

1.

| TOKEN      | CODE | VALUE            |
|------------|------|------------------|
| begin      | 1    | -                |
| end        | 2    | -                |
| if         | 3    | -                |
| else       | 4    | -                |
| then       | 5    | -                |
| identifier | 6    | (6, sym_tab_ptr) |
| constant   | 7    | (7, sym_tab_ptr) |
| ==         | 8    | (8, 1)           |
| !=         | 8    | (8, 2)           |
| >=         | 8    | (8, 3)           |
| <=         | 8    | (8, 4)           |
| <          | 8    | (8, 5)           |
| >          | 8    | (8, 6)           |

Design a DFA that will identify and display the code and values for valid tokens, manage symbol table for identifiers and constants, and find errors.

**Program:**

```
/* Cpp program to detect identifiers and other lexemes like
constants and generate corresponding code or token and
using hash table for symbol table management for lookup and
insert operations in O(1) amortized time */
#include <iostream>
#include <vector>
#include <cstdio>
#include <string>
#define SIZE 100 //Size of hash table
using namespace std;

//const int x = 23; //For hashing
const int x = 19;

struct value{
    int cd; //For corresponding code of token
    string var;
};
struct symbol{
    int e_no; //Entry number in symbol table
```



```

    string name;
    string type;
    value *val;
    symbol **s_table_ptr;//For returning on successful
insertion or lookup say like symbol *sym; s_table_ptr =
&sym;
    symbol *next;//For hashing and chaining
};

symbol *s_table[SIZE] = {NULL}; //Symbol table initialized

symbol *insert(symbol *s_table[SIZE], int ind, int code,
string st, string t)
{
    symbol *newsym = new symbol;
    newsym->e_no = ind;
    newsym->name = t;
    if(code==6)
        newsym->type = "identifier";
    else
        newsym->type = "constant";
    newsym->val = new value;
    newsym->val->cd = code;
    newsym->val->var = st;
    newsym->s_table_ptr = &newsym;
    newsym->next = NULL;
    if(s_table[ind]==NULL)
    {
        s_table[ind] = newsym;
    }
    else
    {
        symbol *s = s_table[ind];
        while(s->next!=NULL)
        {
            s = s->next;
        }
        s->next = newsym;
    }
    return newsym;
}

symbol *look_up(symbol *s_table[SIZE], int ind, int code,

```

```

string st, string t)
{
    symbol *sym = NULL;
    if(s_table[ind]!=NULL)
    {
        symbol *s = s_table[ind];
        while(s!=NULL)
        {
            if((s->name==t) && (s->val->cd==code) &&
(s->val->var==st))
            {
                sym = s;
                break;
            }
            s = s->next;
        }
    }
    return sym;
}

int hashVal(string s)
{
    int n = s.length();
    int h = ((int)s[n-1])%SIZE;
    for(int i=n-2; i>=0; i--)
    {
        h = ((h*x)+s[i])%SIZE;
    }
    return h;
}

int dfa_begin(string s, int n)
{
    int res = 0;
    int s_state = 0, e_state = 5, d_state = 6, c_state =
0, n_state = 0;
    cout<<"In begin DFA\n";
    cout<<"\nStart state no. = "<<s_state<<", end_state =
"<<e_state<<" and, dead state = "<<d_state<<"\n";
    int i = 0, f = 0;
    for(i=0; i<n; i++)
    {
        //printf("current state: %d\n", c_state);
        if(c_state==0 && s[i]=='b')

```

```

        {
            n_state = 1;
        }
        else if(c_state==1 && s[i]=='e')
        {
            n_state = 2;
        }
        else if(c_state==2 && s[i]=='g')
        {
            n_state = 3;
        }
        else if(c_state==3 && s[i]=='i')
        {
            n_state = 4;
        }
        else if(c_state==4 && s[i]=='n')
        {
            n_state = 5;
        }
        else
        {
            n_state = 6;
            f = 1;
            //printf("Invalid input reached dead state
from state no = %d\n", c_state);
            break;
        }
        cout<<"Reached state = "<<n_state<<" from
current_state = "<<c_state<<"\n";
        c_state = n_state;
    }
    if(f==0)
    {
        res = 1;
        cout<<"Valid input keyword begin!!! with code =
"<<res<<"\n";
    }
    else
    {
        cout<<"Not a valid keyword begin!!! reached dead
state = "<<n_state<<" from current state =
"<<c_state<<"\n";
    }
}

```

```

    }
    return res;
}
int dfa_end(string s, int n)
{
    int res = 0;
    int s_state = 0, e_state = 3, d_state = 4, c_state =
0, n_state = 0;
    cout<<"In end DFA\n";
    cout<<"\nStart state no. = "<<s_state<<", end_state =
"<<e_state<<" and, dead state = "<<d_state<<"\n";
    int i = 0, f = 0;
    for(i=0; i<n; i++)
    {
        //printf("current state: %d\n", c_state);
        if(c_state==0 && s[i]=='e')
        {
            n_state = 1;
        }
        else if(c_state==1 && s[i]=='n')
        {
            n_state = 2;
        }
        else if(c_state==2 && s[i]=='d')
        {
            n_state = 3;
        }
        else
        {
            n_state = 4;
            f = 1;
            //printf("Invalid input reached dead state
from state no = %d\n", c_state);
            break;
        }
        cout<<"Reached state = "<<n_state<<" from
current_state = "<<c_state<<"\n";
        c_state = n_state;
    }
    if(f==0)
    {
        res = 2;
    }
}

```

```

        cout<<"Valid input keyword end!!! with code =
"<<res<<"\n";
    }
    else
    {
        cout<<"Not a valid keyword end!!! reached dead
state = "<<n_state<<" from current state =
"<<c_state<<"\n";
    }
    return res;
}
int dfa_if(string s, int n)
{
    int res = 0;
    int s_state = 0, e_state = 2, d_state = 3, c_state =
0, n_state = 0;
    cout<<"In if DFA\n";
    cout<<"\nStart state no. = "<<s_state<<", end_state =
"<<e_state<<" and, dead state = "<<d_state<<"\n";
    int i = 0, f = 0;
    for(i=0; i<n; i++)
    {
        //printf("current state: %d\n", c_state);
        if(c_state==0 && s[i]=='i')
        {
            n_state = 1;
        }
        else if(c_state==1 && s[i]=='f')
        {
            n_state = 2;
        }
        else
        {
            n_state = 3;
            f = 1;
            //printf("Invalid input reached dead state
from state no = %d\n", c_state);
            break;
        }
        cout<<"Reached state = "<<n_state<<" from
current_state = "<<c_state<<"\n";
        c_state = n_state;
    }
}

```

```

    }
    if(f==0)
    {
        res = 3;
        cout<<"Valid input keyword if!!! with code =
"<<res<<"\n";
    }
    else
    {
        cout<<"Not a valid keyword if!!! reached dead
state = "<<n_state<<" from current state =
"<<c_state<<"\n";
    }
    return res;
}
int dfa_else(string s, int n)
{
    int res = 0;
    int s_state = 0, e_state = 4, d_state = 5, c_state =
0, n_state = 0;
    cout<<"In else DFA\n";
    cout<<"\nStart state no. = "<<s_state<<", end_state =
"<<e_state<<" and, dead state = "<<d_state<<"\n";
    int i = 0, f = 0;
    for(i=0; i<n; i++)
    {
        //printf("current state: %d\n", c_state);
        if(c_state==0 && s[i]=='e')
        {
            n_state = 1;
        }
        else if(c_state==1 && s[i]=='l')
        {
            n_state = 2;
        }
        else if(c_state==2 && s[i]=='s')
        {
            n_state = 3;
        }
        else if(c_state==3 && s[i]=='e')
        {
            n_state = 4;

```

```

    }
    else
    {
        n_state = 5;
        f = 1;
        //printf("Invalid input reached dead state
from state no = %d\n", c_state);
        break;
    }
    cout<<"Reached state = "<<n_state<<" from
current_state = "<<c_state<<"\n";
    c_state = n_state;
}
if(f==0)
{
    res = 4;
    cout<<"Valid input keyword else!!! with code =
"<<res<<"\n";
}
else
{
    cout<<"Not a valid keyword else!!! reached dead
state = "<<n_state<<" from current state =
"<<c_state<<"\n";
}
return res;
}
int dfa_then(string s, int n)
{
    int res = 0;
    int s_state = 0, e_state = 4, d_state = 5, c_state =
0, n_state = 0;
    cout<<"In then DFA\n";
    cout<<"\nStart state no. = "<<s_state<<", end_state =
"<<e_state<<" and, dead state = "<<d_state<<"\n";
    int i = 0, f = 0;
    for(i=0; i<n; i++)
    {
        //printf("current state: %d\n", c_state);
        if(c_state==0 && s[i]=='t')
        {
            n_state = 1;

```

```

    }
    else if(c_state==1 && s[i]=='h')
    {
        n_state = 2;
    }
    else if(c_state==2 && s[i]=='e')
    {
        n_state = 3;
    }
    else if(c_state==3 && s[i]=='n')
    {
        n_state = 4;
    }
    else
    {
        n_state = 5;
        f = 1;
        //printf("Invalid input reached dead state
from state no = %d\n", c_state);
        break;
    }
    cout<<"Reached state = "<<n_state<<" from
current_state = "<<c_state<<"\n";
    c_state = n_state;
}
if(f==0)
{
    res = 5;
    cout<<"Valid input keyword then!!! with code =
"<<res<<"\n";
}
else
{
    cout<<"Not a valid keyword then!!! reached dead
state = "<<n_state<<" from current state =
"<<c_state<<"\n";
}
return res;
}
int dfa_rel(string s, int n, int &val)
{
    int res = 0;

```



```

    int s_state = 0, e_state1 = 2, e_state2 = 3, d_state
= 4, c_state = 0, n_state = 0;
    cout<<"In relational operator DFA\n";
    cout<<"\nStart state no. = "<<s_state<<", end_states
are = "<<e_state1<<" and "<<e_state2<<", dead state =
"<<d_state<<"\n";
    int i = 0, f = 0;
    for(i=0; i<n; i++)
    {
        //printf("current state: %d\n", c_state);
        if(c_state==0)
        {
            if(s[i]=='>')
            {
                n_state = 3;
                val = 6;
            }
            else if(s[i]=='<')
            {
                n_state = 3;
                val = 5;
            }
            else if(s[i]=='!')
            {
                n_state = 3;
                val = 2;
            }
            else
            {
                if(s[i]=='=')
                {
                    n_state = 1;
                    val = 1;
                }
                else
                {
                    n_state = 4;
                    f = 1;
                    val = -1;
                    break;
                }
            }
        }
    }

```

```

    }
    else if(c_state==1)
    {
        if(s[i]=='=')
            n_state = 2;
        else
        {
            n_state = 4;
            f = 1;
            val = -1;
            break;
        }
    }
    else if(c_state==2)
    {
        if(i<n)//Means there are still other
symbols of string left to be processed
        {
            n_state = 4;
            f = 1;
            val = -1;
            break;
        }
        else
        {
            n_state = 2;
            break;
        }
    }
    else if(c_state==3)
    {
        if(s[i]=='=')//Means there are still other
symbols of string left to be processed
        {
            n_state = 2;
            if(val==6)
            {
                val = 3;
            }
            else if(val==5)
            {
                val = 4;
            }
        }
    }

```

```

        }
    }
    else
    {
        n_state = 4;
        f = 1;
        val = -1;
        break;
    }
}
else
{
    n_state = 4;
    f = 1;
    val = -1;
    //printf("Invalid input reached dead state
from state no = %d\n", c_state);
    break;
}
cout<<"Reached state = "<<n_state<<" from
current_state = "<<c_state<<"\n";
c_state = n_state;
}
if(f==0)
{
    res = 8;
    cout<<"Valid input relational operator!!! with
code = "<<res<<" and value = "<<val<<"\n";
}
else
{
    cout<<"Not a valid relational operator!!!
reached dead state = "<<n_state<<" from current state =
"<<c_state<<"\n";
}
return res;
}
int dfa_iden(string s, int n)
{
    int res = 0;
    vector<char> st;
    //For int

```

```

    int s_state = 0, e_state = 5, d_state = 7, c_state =
0, n_state = 0, str = 0; //str used to count no of * symbols
    cout<<"In int DFA\n";
    cout<<"\nStart state no. = "<<s_state<<", end_state =
"<<e_state<<" and, dead state = "<<d_state<<"\n";
    int i = 0, f = 0;
    for(i=0; i<n; i++)
    {
        //printf("current state: %d\n", c_state);
        if(c_state==0)
        {
            if(s[i]=='i')
                n_state = 1;
            else
            {
                n_state = 7;
                f = 1;
                break;
            }
        }
        else if(c_state==1)
        {
            if(s[i]=='n')
                n_state = 2;
            else
            {
                n_state = 7;
                f = 1;
                break;
            }
        }
        else if(c_state==2)
        {
            if(s[i]=='t')
                n_state = 3;
            else
            {
                n_state = 7;
                f = 1;
                break;
            }
        }
    }
}

```

```

else if(c_state==3)
{
    if(s[i]==' ')
        n_state = 4;
    else if(s[i]=='*')
    {
        str++;
        //st.push_back(s[i]);
        n_state = 6;
    }
    else
    {
        n_state = 7;
        f = 1;
        break;
    }
}
else if(c_state==4)
{
    if(s[i]==' ')
    {
        n_state = 4;
    }
    else if(s[i]=='*')
    {
        str++;
        //st.push_back(s[i]);
        n_state = 6;
    }
    else if((((int)(s[i])>=65) && ((int)
(s[i])<=90)) || (((int)(s[i])>=97) && ((int)(s[i])<=122))
|| (s[i]=='_'))
    {
        st.push_back(s[i]);
        n_state = 5;//Reached end state
    }
    else
    {
        n_state = 7;
        f = 1;
        break;
    }
}

```

```

    }
    else if(c_state==5)
    {
        if((((int)(s[i])>=65) && ((int)(s[i])<=90))
|| (((int)(s[i])>=97) && ((int)(s[i])<=122)) || (s[i]=='_')
|| (((int)(s[i])>=48) && ((int)(s[i])<=57))))
        {
            st.push_back(s[i]);
            n_state = 5;//Reached end state
        }
        else
        {
            n_state =7;
            f = 1;
            break;
        }
    }
    else if(c_state==6)
    {
        if(s[i]==' ')
        {
            n_state = 4;
        }
        else if(s[i]=='*')
        {
            str++;
            //st.push_back(s[i]);
            n_state = 6;
        }
        else if((((int)(s[i])>=65) && ((int)
(s[i])<=90)) || (((int)(s[i])>=97) && ((int)(s[i])<=122))
|| (s[i]=='_'))
        {
            st.push_back(s[i]);
            n_state = 5;//Reached end state
        }
        else
        {
            n_state = 7;
            f = 1;
            break;
        }
    }

```

```

    }
    cout<<"Reached state = "<<n_state<<" from
current_state = "<<c_state<<"\n";
    c_state = n_state;
}
string id1 = "int";
while(str--)
{
    id1 += "*";
}
if(f==0)
{
    if(st.size()==0)//To check if there is any null
symbol after int or int * or int***** etc which is invalid
input/syntax
    {
        n_state = 7;//Reached dead state
        cout<<"Not a valid identifier of type
"<<id1<<". Missing variable name!!! reached dead state =
"<<n_state<<" from current state = "<<c_state<<"\n";
        res = -1;
    }
    else
    {
        /*int n = st.size();
char vr[n+1];
for(int i=0; i<n; i++)
{
    vr[i] = st[i];
}
vr[n] = '\0';*/
//int sz = n+1;
string vr(st.begin(), st.end());
if((vr!="begin")&&(vr!="end")&&(vr!
="if")&&(vr!="else")&&(vr!="then")&&((vr!="int") && (vr!
="char"))&&((vr!="0")&&(vr!="1")&&(vr!="2")&&(vr!
="3")&&(vr!="4")&&(vr!="5")&&(vr!="6")&&(vr!="7")&&(vr!
="8")&&(vr!="9"))&&((vr!="=")&&(vr!="!")&&(vr!
="<=")&&(vr!=">=")&&(vr!="<")&&(vr!=">"))
    {
        res = 6;
        cout<<"Valid input identifier

```

```

"<<vr<<" of type "<<id1<<"!!! with code = "<<res<<"\n";
        //cout<<vr<<"\n";
        //cout<<id1<<"\n";
        int hint = hashVal(id1);
        int h = (hint+hashVal(vr))%SIZE;
        symbol *sym = look_up(s_table, h,
res, vr, id1);
        if(sym!=NULL)
        {
            cout<<"Variable "<<vr<<" already
declared at location ("<<"symbol_table_enty_no = "<<sym-
>e_no<<", symbol_table_ptr_addr = "<<sym->s_table_ptr<<")
with name = "<<sym->name<<", type = "<<sym->type<<", and
value (code = "<<sym->val->cd<<", variable name = "<<sym-
>val->var<<")\n";
        }
        else
        {
            //cout<<"Not found\n";
            symbol *ins = insert(s_table, h,
res, vr, id1); //res = code for int token here
            cout<<"Variable "<<vr<<"
inserted at location ("<<"symbol_table_enty_no = "<<ins-
>e_no<<", symbol_table_ptr_addr = "<<ins->s_table_ptr<<")
with name = "<<ins->name<<", type = "<<ins->type<<", and
value (code = "<<ins->val->cd<<", variable name = "<<ins-
>val->var<<")\n";
        }
    }
    else
    {
        cout<<"Invalid input identifier
"<<vr<<" of type "<<id1<<" as "<<vr<<" is a reserved
keyword\n";
        res = -1;
    }
}
}
else
{
    cout<<"Not a valid identifier of type
"<<id1<<"!!! reached dead state = "<<n_state<<" from

```



```

current state = "<<c_state<<"\n";

/*----- FOR CHAR
-----*/
//For char
st.clear();
s_state = 0, e_state = 6, d_state = 8, c_state =
0, n_state = 0;
cout<<"In char DFA\n";
cout<<"\nStart state no. = "<<s_state<<",
end_state = "<<e_state<<" and, dead state =
"<<d_state<<"\n";
i = 0, f = 0, str = 0, id1 = "char";
for(i=0; i<n; i++)
{
    //printf("current state: %d\n", c_state);
    if(c_state==0)
    {
        if(s[i]=='c')
            n_state = 1;
        else
        {
            n_state = 8;
            f = 1;
            break;
        }
    }
    else if(c_state==1)
    {
        if(s[i]=='h')
            n_state = 2;
        else
        {
            n_state = 8;
            f = 1;
            break;
        }
    }
    else if(c_state==2)
    {
        if(s[i]=='a')
            n_state = 3;
    }
}

```

```

        else
        {
            n_state = 8;
            f = 1;
            break;
        }
    }
else if(c_state==3)
{
    if(s[i]=='r')
        n_state = 4;
    else
    {
        n_state = 8;
        f = 1;
        break;
    }
}
else if(c_state==4)
{
    if(s[i]==' ')
        n_state = 5;
    else if(s[i]=='*')
    {
        str++;
        n_state = 7;
    }
    else
    {
        n_state = 8;
        f = 1;
        break;
    }
}
else if(c_state==5)
{
    if(s[i]==' ')
    {
        n_state = 5;
    }
    else if(s[i]=='*')
    {

```

```

        str++;
        n_state = 7;
    }
    else if((((int)(s[i])>=65) && ((int)
(s[i])<=90)) || (((int)(s[i])>=97) && ((int)(s[i])<=122))
|| (s[i]=='_'))
    {
        st.push_back(s[i]);
        n_state = 6;//Reached end
state
    }
    else
    {
        n_state = 8;
        f = 1;
        break;
    }
}
else if(c_state==6)
{
    if((((int)(s[i])>=65) && ((int)
(s[i])<=90)) || (((int)(s[i])>=97) && ((int)(s[i])<=122))
|| (s[i]=='_') || (((int)(s[i])>=48) && ((int)(s[i])<=57)))
    {
        st.push_back(s[i]);
        n_state = 6;//Reached end state
    }
    else
    {
        n_state =8;
        f = 1;
        break;
    }
}
else if(c_state==7)
{
    if(s[i]==' ')
    {
        n_state = 5;
    }
    else if(s[i]=='*')
    {

```

```

        str++;
        n_state = 7;
    }
    else if((((int)(s[i])>=65) && ((int)
(s[i])<=90)) || (((int)(s[i])>=97) && ((int)(s[i])<=122))
|| (s[i]=='_'))
    {
        st.push_back(s[i]);
        n_state = 6;//Reached end state
    }
    else
    {
        n_state = 8;
        f = 1;
        break;
    }
}
cout<<"Reached state = "<<n_state<<" from
current_state = "<<c_state<<"\n";
c_state = n_state;
}
while(str--)
{
    id1 += "*";
}
if(f==0)
{
    if(st.size()==0)
    {
        n_state = 8;
        cout<<"Not a valid identifier of type
"<<id1<<". Missing variable name!!! reached dead state =
"<<n_state<<" from current state = "<<c_state<<"\n";
        res = -1;
    }
    else
    {
        string vr(st.begin(), st.end());
        if((vr!="begin")&&(vr!="end")&&(vr!
="if")&&(vr!="else")&&(vr!="then")&&((vr!="int") && (vr!
="char"))&&((vr!="0")&&(vr!="1")&&(vr!="2")&&(vr!
="3")&&(vr!="4")&&(vr!="5")&&(vr!="6")&&(vr!="7")&&(vr!

```

```

="8")&&(vr!="9"))&&((vr!="==")&&(vr!="!=")&&(vr!
="<=")&&(vr!=">=")&&(vr!="<")&&(vr!=">"))
    {
        res = 6;
        cout<<"Valid input identifier
"<<vr<<" of type "<<id1<<"!!! with code = "<<res<<"\n";
        //cout<<vr<<"\n";
        //cout<<id1<<"\n";
        int hchar = hashVal(id1);
        int h = (hchar+hashVal(vr))

%SIZE;

        symbol *sym = look_up(s_table,
h, res, vr, id1);

        if(sym!=NULL)
        {
            cout<<"Variable "<<vr<<"
already declared at location ("<<"symbol_table_enty_no =
"<<sym->e_no<<", symbol_table_ptr_addr = "<<sym-
>s_table_ptr<<") with name = "<<sym->name<<", type =
"<<sym->type<<", and value (code = "<<sym->val->cd<<",
variable name = "<<sym->val->var<<")\n";
        }
        else
        {
            //cout<<"Not found\n";
            symbol *ins =
insert(s_table, h, res, vr, id1); //res = code for int token
here

            cout<<"Variable "<<vr<<"
inserted at location ("<<"symbol_table_enty_no = "<<ins-
>e_no<<", symbol_table_ptr_addr = "<<ins->s_table_ptr<<")
with name = "<<ins->name<<", type = "<<ins->type<<", and
value (code = "<<ins->val->cd<<", variable name = "<<ins-
>val->var<<")\n";
        }
    }
    else
    {
        cout<<"Invalid input identifier
"<<vr<<" of type "<<id1<<" as "<<vr<<" is a reserved
keyword\n";

        res = -1;
    }
}

```

```

        }
    }
}
else
{
    cout<<"Not a valid identifier of type
"<<id1<<"!!! reached dead state = "<<n_state<<" from
current state = "<<c_state<<"\n";
}
}
return res;
}
int dfa_cons(string s, int n)
{
    int res = 0;
    int s_state = 0, e_state = 1, d_state = 2, c_state =
0, n_state = 0;
    cout<<"In constant DFA\n";
    cout<<"\nStart state no. = "<<s_state<<", end_state =
"<<e_state<<" and, dead state = "<<d_state<<"\n";
    int i = 0, f = 0;
    string id2 = "number";
    vector<char > st;
    for(i=0; i<n; i++)
    {
        //printf("current state: %d\n", c_state);
        if(c_state==0)
        {
            if(((int)(s[i])>=48) && ((int)(s[i])<=57))
            {
                st.push_back(s[i]);
                n_state = 1;
            }
            else
            {
                n_state = 2;
                f = 1;
                break;
            }
        }
        else if(c_state==1)
        {

```

```

        if(((int)(s[i])>=48) && ((int)(s[i])<=57))
        {
            st.push_back(s[i]);
            n_state = 1;
        }
        else
        {
            n_state = 2;
            f = 1;
            break;
        }
    }
    else
    {
        n_state = 2;
        f = 1;
        //printf("Invalid input reached dead state
from state no = %d\n", c_state);
        break;
    }
    cout<<"Reached state = "<<n_state<<" from
current_state = "<<c_state<<"\n";
    c_state = n_state;
}
if(f==0)
{
    if(st.size()==0)
    {
        n_state = 2;
        cout<<"Not a valid identifier of type
"<<id2<<". Missing number!!! reached dead state =
"<<n_state<<" from current state = "<<c_state<<"\n";
        res = -1;
    }
    else
    {
        res = 7;
        cout<<"Valid input constant of type
"<<id2<<"!!! with code = "<<res<<"\n";
        int hcons = hashVal(id2);
        int h = (hcons+hashVal(s))%SIZE;
        symbol *sym = look_up(s_table, h, res, s,

```

```

id2);
        if(sym!=NULL)
        {
            cout<<"Constant "<<s<<" already
declared at location ("<<"symbol_table_enty_no = "<<sym-
>e_no<<", symbol_table_ptr_addr = "<<sym->s_table_ptr<<")
with name = "<<sym->name<<", type = "<<sym->type<<", and
value (code = "<<sym->val->cd<<", variable name = "<<sym-
>val->var<<")\n";
        }
        else
        {
            //cout<<"Not found\n";
            symbol *ins = insert(s_table, h, res,
s, id2); //res = code for int token here
            cout<<"Constant "<<s<<" inserted at
location ("<<"symbol_table_enty_no = "<<ins->e_no<<",
symbol_table_ptr_addr = "<<ins->s_table_ptr<<") with name =
"<<ins->name<<", type = "<<ins->type<<", and value (code =
"<<ins->val->cd<<", variable name = "<<ins->val-
>var<<")\n";
        }
    }
    else
    {
        cout<<"Not a valid constant of type "<<id2<<"!!!
reached dead state = "<<n_state<<" from current state =
"<<c_state<<"\n";
    }
    return res;
}
void dfa(string s)
{
    int code = 0; //Initialized
    int val = -1;
    int n = s.length();
    if(code==0)
    {
        code = dfa_begin(s, n);
        if(code==0)
        {

```



```

        code = dfa_end(s, n);
        if(code==0)
        {
            code = dfa_if(s, n);
            if(code==0)
            {
                code = dfa_else(s, n);
                if(code==0)
                {
                    code = dfa_then(s, n);
                    if(code==0)
                    {
                        code = dfa_iden(s,
n);
                        if(code==0)
                        {
                            code =
dfa_cons(s, n);
                            if(code==0)
                            {
                                code =
dfa_rel(s, n, val);
                                if(code==0)
                                {
                                    cout<<"Invalid input please enter a valid token and
try again!!!\n";
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
int main()
{

```

```

string line;
int Q;
cout<<"Enter number of queries: ";
cin>>Q;
getchar();
while(Q--)
{
    cout<<"Enter the tokens \n";
    getline(cin, line);
    cout<<"You entered "<<line<<"\n";
    dfa(line);
}
return 0;
}

```

```

ntu: ~/compiler_lab
dt@ubuntu:~/compiler_lab$ g++ p4_1final.cpp
dt@ubuntu:~/compiler_lab$ ./a.out
Enter number of queries: 4
Enter the tokens
int a
You entered int a
In begin DFA

Start state no. = 0, end_state = 5 and, dead state = 6
Not a valid keyword begin!!! reached dead state = 6 from current state = 0
In end DFA

Start state no. = 0, end_state = 3 and, dead state = 4
Not a valid keyword end!!! reached dead state = 4 from current state = 0
In if DFA

Start state no. = 0, end_state = 2 and, dead state = 3
Reached state = 1 from current_state = 0
Not a valid keyword if!!! reached dead state = 3 from current state = 1
In else DFA

Start state no. = 0, end_state = 4 and, dead state = 5
Not a valid keyword else!!! reached dead state = 5 from current state = 0
In then DFA

Start state no. = 0, end_state = 4 and, dead state = 5
Not a valid keyword then!!! reached dead state = 5 from current state = 0
In int DFA

Start state no. = 0, end_state = 5 and, dead state = 7
Reached state = 1 from current_state = 0
Reached state = 2 from current_state = 1
Reached state = 3 from current_state = 2
Reached state = 4 from current_state = 3
Reached state = 5 from current_state = 4
Valid input identifier a of type int!!! with code = 6
Variable a inserted at location (symbol_table_enty_no = 68, symbol_table_ptr_add
r = 0x7fff36012540) with name = int, type = identifier, and value (code = 6, var
iable name = a)
Enter the tokens

```

```

ntu: ~/compiler_lab
table name = a)
Enter the tokens
int * *a
You entered int * *a
In begin DFA

Start state no. = 0, end_state = 5 and, dead state = 6
Not a valid keyword begin!!! reached dead state = 6 from current state = 0
In end DFA

Start state no. = 0, end_state = 3 and, dead state = 4
Not a valid keyword end!!! reached dead state = 4 from current state = 0
In if DFA

Start state no. = 0, end_state = 2 and, dead state = 3
Reached state = 1 from current_state = 0
Not a valid keyword if!!! reached dead state = 3 from current state = 1
In else DFA

Start state no. = 0, end_state = 4 and, dead state = 5
Not a valid keyword else!!! reached dead state = 5 from current state = 0
In then DFA

Start state no. = 0, end_state = 4 and, dead state = 5
Not a valid keyword then!!! reached dead state = 5 from current state = 0
In int DFA

Start state no. = 0, end_state = 5 and, dead state = 7
Reached state = 1 from current_state = 0
Reached state = 2 from current_state = 1
Reached state = 3 from current_state = 2
Reached state = 4 from current_state = 3
Reached state = 6 from current_state = 4
Reached state = 4 from current_state = 6
Reached state = 6 from current_state = 4
Reached state = 5 from current_state = 6
Valid input identifier a of type int**!!! with code = 6
Variable a inserted at location (symbol_table_enty_no = 28, symbol_table_ptr_addr = 0x7fff36012540) with name = int**, type = identifier, and va
lue (code = 6, variable name = a)
Enter the tokens

```

```

ntu: ~/compiler_lab
Reached state = 5 from current_state = 6
Valid input identifier a of type int**!!! with code = 6
Variable a inserted at location (symbol_table_enty_no = 28, symbol_table_ptr_addr = 0x7fff36012540) with name = int**, type = identifier, and va
lue (code = 6, variable name = a)
Enter the tokens
int a
You entered int a
In begin DFA

Start state no. = 0, end_state = 5 and, dead state = 6
Not a valid keyword begin!!! reached dead state = 6 from current state = 0
In end DFA

Start state no. = 0, end_state = 3 and, dead state = 4
Not a valid keyword end!!! reached dead state = 4 from current state = 0
In if DFA

Start state no. = 0, end_state = 2 and, dead state = 3
Reached state = 1 from current_state = 0
Not a valid keyword if!!! reached dead state = 3 from current state = 1
In else DFA

Start state no. = 0, end_state = 4 and, dead state = 5
Not a valid keyword else!!! reached dead state = 5 from current state = 0
In then DFA

Start state no. = 0, end_state = 4 and, dead state = 5
Not a valid keyword then!!! reached dead state = 5 from current state = 0
In int DFA

Start state no. = 0, end_state = 5 and, dead state = 7
Reached state = 1 from current_state = 0
Reached state = 2 from current_state = 1
Reached state = 3 from current_state = 2
Reached state = 4 from current_state = 3
Reached state = 5 from current_state = 4
Valid input identifier a of type int!!! with code = 6
Variable a already declared at location (symbol_table_enty_no = 68, symbol_table_ptr_addr = 0x7fff36012540) with name = int, type = identifier,
and value (code = 6, variable name = a)
Enter the tokens

```

```

dt@ubuntu:~/compiler_lab
Enter the tokens
900
You entered 900
In begin DFA

Start state no. = 0, end_state = 5 and, dead state = 6
Not a valid keyword begin!!! reached dead state = 6 from current state = 0
In end DFA

Start state no. = 0, end_state = 3 and, dead state = 4
Not a valid keyword end!!! reached dead state = 4 from current state = 0
In if DFA

Start state no. = 0, end_state = 2 and, dead state = 3
Not a valid keyword if!!! reached dead state = 3 from current state = 0
In else DFA

Start state no. = 0, end_state = 4 and, dead state = 5
Not a valid keyword else!!! reached dead state = 5 from current state = 0
In then DFA

Start state no. = 0, end_state = 4 and, dead state = 5
Not a valid keyword then!!! reached dead state = 5 from current state = 0
In int DFA

Start state no. = 0, end_state = 5 and, dead state = 7
Not a valid identifier of type int!!! reached dead state = 7 from current state = 0
In char DFA

Start state no. = 0, end_state = 6 and, dead state = 8
Not a valid identifier of type char!!! reached dead state = 8 from current state = 0
In constant DFA

Start state no. = 0, end_state = 1 and, dead state = 2
Reached state = 1 from current_state = 0
Reached state = 1 from current_state = 1
Reached state = 1 from current_state = 1
Valid input constant of type number!!! with code = 7
Constant 900 inserted at location (symbol_table_ptr_addr = 0x7fff360125c0) with name = number, type = constant, and v
alue (code = 7, variable name = 900)
dt@ubuntu:~/compiler_lab$

```

```

dt@ubuntu:~/compiler_lab
value (code = 7, variable name = 900)
dt@ubuntu:~/compiler_lab$ ./a.out
Enter number of queries: 2
Enter the tokens
begin
You entered begin
In begin DFA

Start state no. = 0, end_state = 5 and, dead state = 6
Reached state = 1 from current_state = 0
Reached state = 2 from current_state = 1
Reached state = 3 from current_state = 2
Reached state = 4 from current_state = 3
Reached state = 5 from current_state = 4
Valid input keyword begin!!! with code = 1
Enter the tokens
>=
You entered >=
In begin DFA

Start state no. = 0, end_state = 5 and, dead state = 6
Not a valid keyword begin!!! reached dead state = 6 from current state = 0
In end DFA

Start state no. = 0, end_state = 3 and, dead state = 4
Not a valid keyword end!!! reached dead state = 4 from current state = 0
In if DFA

Start state no. = 0, end_state = 2 and, dead state = 3
Not a valid keyword if!!! reached dead state = 3 from current state = 0
In else DFA

Start state no. = 0, end_state = 4 and, dead state = 5
Not a valid keyword else!!! reached dead state = 5 from current state = 0
In then DFA

Start state no. = 0, end_state = 4 and, dead state = 5
Not a valid keyword then!!! reached dead state = 5 from current state = 0
In int DFA

Start state no. = 0, end_state = 5 and, dead state = 7

```

untu: ~/compiler\_lab

4:21 PM

```
>=
You entered >=
In begin DFA

Start state no. = 0, end_state = 5 and, dead state = 6
Not a valid keyword begin!!! reached dead state = 6 from current state = 0
In end DFA

Start state no. = 0, end_state = 3 and, dead state = 4
Not a valid keyword end!!! reached dead state = 4 from current state = 0
In if DFA

Start state no. = 0, end_state = 2 and, dead state = 3
Not a valid keyword if!!! reached dead state = 3 from current state = 0
In else DFA

Start state no. = 0, end_state = 4 and, dead state = 5
Not a valid keyword else!!! reached dead state = 5 from current state = 0
In then DFA

Start state no. = 0, end_state = 4 and, dead state = 5
Not a valid keyword then!!! reached dead state = 5 from current state = 0
In int DFA

Start state no. = 0, end_state = 5 and, dead state = 7
Not a valid identifier of type int!!! reached dead state = 7 from current state = 0
In char DFA

Start state no. = 0, end_state = 6 and, dead state = 8
Not a valid identifier of type char!!! reached dead state = 8 from current state = 0
In constant DFA

Start state no. = 0, end_state = 1 and, dead state = 2
Not a valid constant of type number!!! reached dead state = 2 from current state = 0
In relational operator DFA

Start state no. = 0, end_states are = 2 and 3, dead state = 4
Reached state = 3 from current_state = 0
Reached state = 2 from current_state = 3
Valid input relational operator!!! with code = 8 and value = 3
dt@ubuntu:~/compiler_lab$
```

**Date: 30-01-2018**

1. Design a flex bison application that will check the correctness of declaration of variables and functions in C language.

Example:

type id1, id2;

type func1 ( type param1, type param2);

type = {void, int, char, float} (keywords)

id1, id2, func1, param1, param2 are identifiers.

Token separators or punctuations are , ; ( )

**Program:**

p5\_1.l

/\* Flex and Bison application to check valid identifier and function declaration as in C \*/

%{

#include "p5\_1.tab.h"

%}

%%

("int")[ ]|("float")[ ]|("char")[ ]|("void")[ ] { return KEY;}

[a-zA-Z\_][a-zA-Z0-9]\* { return ID; }

;" { return END;}

," { return COMMA;}

(" { return LP;}

")" { return RP;}

[\n] { return EOL;}

[ \t] { }

. { printf("Invalid Identifier\n");  
return I;}

%%

p5\_1.y

/\* bison program for checking correctness of declaration of variables and functions in C language \*/

%{

#include <stdio.h>

int yylex();

void yyerror(const char \*s);

%}

```

%token KEY
%token ID
%token END
%token COMMA
%token LP
%token RP
%token EOL
%token I
%%
S:
| KEY ID LP chk RP END S { printf(" Correct function
declaration!! \n"); return 0;}
| KEY ID LP chk RP END EOL { printf(" Correct function
declaration!! \n"); return 0;}
| KEY iden END S { printf(" Correct identifier
declaration!! \n"); return 0;}
| KEY iden END EOL{ printf(" Correct identifier
declaration!! \n"); return 0;}
| KEY EOL{printf("Incorrect\n Incomplete Statement \n");
return 0;}
| KEY iden EOL{printf("Incorrect\n Incomplete Statement \n"
); return 0;}
| KEY inv EOL{printf("Incorrect\n Incomplete Statement
\n" ); return 0;}
;
chk:
| KEY ID { printf("Valid\n"); }
| KEY ID COMMA chk
;
iden: iden COMMA ID { }
| ID { }
;
inv: I { }
;
%%

main(int argc, char **argv) {
    yyparse();
}

void yyerror(const char *s) {
    printf("%s \n", s);
}

```

}

```
dt@ubuntu: ~/compiler_lab/cd_lab_30_1_18
dt@ubuntu:~/compiler_lab/cd_lab_30_1_18$ ./p5_1test
int a
Incorrect
  Incomplete Statement
dt@ubuntu:~/compiler_lab/cd_lab_30_1_18$ ./p5_1test
int a; int fun(int a, void b)
Valid
syntax error
dt@ubuntu:~/compiler_lab/cd_lab_30_1_18$ ./p5_1test
int a; int fun(int a, void b);
Valid
  Correct function declaration!!
dt@ubuntu:~/compiler_lab/cd_lab_30_1_18$ ./p5_1test
int a, b, c;
  Correct identifier declaration!!
dt@ubuntu:~/compiler_lab/cd_lab_30_1_18$ ./p5_1test
void min();
  Correct function declaration!!
dt@ubuntu:~/compiler_lab/cd_lab_30_1_18$ ./p5_1test
float a, b;
  Correct identifier declaration!!
dt@ubuntu:~/compiler_lab/cd_lab_30_1_18$ ./p5_1test
char a
Incorrect
  Incomplete Statement
dt@ubuntu:~/compiler_lab/cd_lab_30_1_18$
```

\*\*\*\*\*