Compiler Lab
List of Experiments

Lab 1. [Day 01]: Introduction to Flex (Lex)

a. Using Flex
b. Conversion of Roman numbers to Decimal numbers using Flex

Lab 2. [Day 02]: Introduction to Flex

a. Identification of various number formats using Flex

Lab 3. [Day 03]: Introduction to Bison (YACC)

a. Using Bison
b. English sentence type parsing with Bison
c. Multiple input process with recursive grammar rules

Lab 4. [Day 04-06]: Make a Calculator using Flex and Bison

a. The calculator should have the following capabilities:
    i. +, -, *, /, ^, unary minus and = operations
    ii. Allow decimal numbers and read input from a File
    iii. Maintain usual associativity and precedence among operators
    iv. Allow parenthesis [(,)] to override precedence
    v. Allow multi-character variable names and maintain variable states throughout a session. Assume that all variables are of type decimal
    vi. Allow repeated inputs and terminate with a single $ symbol
    vii. Allow the following mathematical function SQRT(n), LOG(n) (base2) and EXP(n)

Sample Execution of the Calculator
2/3+1
= 1.666667
(4*(1+2)^3)/6
=18.000000
a = 2 + 6.5 / 3.25
a
= 4.000000
my_variable = SQRT(a) + 6
my_variable/a
= 5.000000
LOG(512.00) * EXP(1.00)
=24.464536

a. Implement the theory detailed in Section 3.9, Page 134-144 in the Compiler book of [Aho, Ullman] using C or C++ (or Java).
b. Represent the empty transition ε, with a capital E character.
c. Regular Expressions are given in decreasing order of precedence.

Sample Execution of the DFA from Regular Expression Generator

Number of Regular Expression
5
aa(b+E)*c**
(d+E)(a+b+E)a+cd
(a+b+c+d+e+f)(pqr)*(xyz)*(a+b)*
*((a+b)(a+b)(a+b)a*b*c*)*b*c**
(a(c+d)+(a(e+f+E)))(a+cd)(e*(f+g))**

Number of Test String
10
E
Accepted by RE 1
dba
Accepted by RE 2
aaa
Accepted by RE 1
frz
Maximally Accepted by RE 3
acadaeafacdefeg
Accepted by RE 5
dcd
Accepted by RE 5
daad
Maximally Accepted by RE 2
bbbbbbbbbbbbbbbbbbbbbbbbbbbb
Accepted by RE 4
abcdefpqrpqrxyzxyz
Maximally Accepted by RE 3
wxyz
Accepted by NONE

a. Theory from Chapter 8 of Compiler book. Specifically Section 8.6 is important.
b. Your code must support all of following type of constructs:
    i. Boolean Expression: [See page 501]
    E -> **E** OP **E** | **E** or **E** | **E** and **E** | not **E** | (**E**) | **id** RELOP **id** | -**E** | ε
    where 'id' is any variable or integer and RELOP is [>, <, ==, !=] and OP is [+,-,*,/]

    ii. Statements: [See page 504]
    S -> **id** = **E** |
    while **E** do **S** |
    do **S** while **E** |
    for (**S** ; **E** ; **S**) **S** |
    if **E** then **S** |
    break |
    continue |
    if **E** then **S** else **S** |
    { **L** }

    iii. For allowing multiple statements: [See page 504] **L** is the starting state.
    L -> **L** ; **S** | **S** | ε

Sample Execution of the Intermediate Code Generator

Input	**Output**

for (a = 1; a < 100; a = a + 1)
{
do
{
if (a > b) or (a != 0) then
{
j = j + 1;
a = a – 1
}
else
break;
v = x * y
}
while ((a < 5) and (v > 10)) ;
a = a + b * 15
};
c = c * d - 7 / 4
$
	100: a = 1
	101: if a < 100 goto 106
	102: goto 126
	103: t1 = a + 1
	104: a = t1
	105: goto 101
	106: if a > b goto 110
	107: goto 108
	108: if a != 0 goto 110
	109: goto 115
	110: t2 = j + 1
	111: j = t2
	112: t3 = a - 1
	113: a = t3
	114: goto 116
	115: goto 122
	116: t4 = x * y
	117: v = t4
	118: if a < 5 goto 120
	119: goto 122
	120: if v > 10 goto 106

	121: goto 122 122: t5 = b * 15 123: t6 = t5 + a 124: a = t6 125: goto 103 126: t7 = c + d 127: t8 = 7 / 4; 128: t9 = t7 - t8 129: c = t9
	* Note that your output may be different from the given sample output. Any **correct** output will be accepted.

Lab 7. [Day xx-xx]: Extension of Lab 6 by Introducing Procedures, Recursive Procedures and Arrays.