Lab Assignment: Compiler Design (COD - 632)

1. Analyze the object code
2. Download and analyze the Lex/Flex tool
3. Write a program to Convert regular expression to DFA directly using *firstpos*(), *laspos*() and *followpos*(). Display the result of *firstpos*(), *laspos*() and *followpos*() before printing the DFA state transition.
4. Write a C/C++/Java program to tokenize the simple "Hello World" program of C language. First of all, define the token for this program. The original program should necessarily include few comments. Display the tokens by removing all the comments, i.e., the comments should be ignored for tokenization. No predefined function for tokenization is allowed. Assume the situations, with and without space between the tokens in the program.

Sample I/P in Python	Sample Lexical output
print (3 + x *2) # comment	(keyword , "print") (delim , "(") (int , 3) (punct, "+") (id , "x") (punct, "*") (int, 2) (delim, ")")

5. Write a Program in Flex which identifies C Integers and float numbers. Your program should respond to various inputs as follows:

Sample I/P	Sample Lexical output
234 -765 8234.01	Integer Integer Float

6. Write a program to eliminate the Left Recursion using the given in Algorithm 4.19 (Page No. 213) of Compilers Principles, Techniques and Tools book.

Sample Grammar	Sample output
E -> E + T | T T -> T * F |F F -> (E) | id	E -> TE' E' -> +TE' | Є T -> FT' T' -> *FT' | Є F -> (E) | id

7. Write a program to find the FIRST for all grammar symbols and FOLLOW for all Non-Terminals in a given grammar.

Sample Grammar	Sample output
E -> TE' E' -> +TE' | Є T -> FT' T' -> *FT' | Є F -> (E) | id	FIRST (E) = {(,id} FIRST (T) = {(,id} FIRST (F) = {(,id} FIRST (E') = {+, Є} FIRST (T') = {*, Є} FIRST (+) = {+} FIRST (*) = {*} FIRST (id) = {id} //optional FIRST (() = {(} FIRST ()) = {)} FOLLOW (E) = {), $} FOLLOW (E') = {), $} FOLLOW (T) = {+,), $} FOLLOW (T') = {+,), $} FOLLOW (F) = {*,+,), $}

Note: Write your own algorithm which can produce FIRST and FOLLOW for any grammar.

8. Write a program to construct the LL(1) parsing table or predictive parsing table using the algorithm given in Algorithm 4.31 (Page No. 224) of Compilers Principles, Techniques and Tools book. Use the grammar and the FIRST & FOLLOW of the previous experiment and construct the table.

Sample I/P Grammar	Sample output
E -> TE' E' -> +TE' | Є T -> FT' T' -> *FT' | Є F -> (E) | id	M[E,id] = E->TE' M[E,(] = E->TE' M[E', +] = E'->+TE' M[E',)] = E'-> Є M[E', $] = E'-> Є M[T,id] = T->FT' M[T,(] = T->FT' M[T',+] = T'-> Є M[T', *] = T'->*FT' M[T',)] = T'-> Є M[T', $] = T'-> Є M[F,id] = F->id M[F,(] = F->(E)

9. Consider following grammar
   S →EF|AF|EB|AB
   X→AY|BY|a|b
   Y→AY|BY|a|b

E→AX
F→BX
A→a
B→b

Write the C/C++/Java program using the CYK algorithm to recognize the strings produced by the above grammar.

Sample String	Sample output
aaa	No
ab	Yes
ababb	Yes

After completing the enough tasks for the above grammar, give your own grammar of the Chomsky Normal Form and check its applicability. [CYK is not discussed in the class. You study on your own and implement it]

10. Write a C/C++/ Java program to implement the LR(0) and SLR(1) parser.
    a. Create the LR(0) item sets
    b. Make LR(0) and SLR(1) table.
    c. Create the parser and call the table according to parser required.
    d. Print the error also.

11. Work on YACC and do the simple parsing and translation.