Lab Assignment: Compiler Design (COD - 632)

1. Analyze the object code
2. Download and analyze the Flex tool
3. 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, ")")

4. Consider following grammar

   $S \rightarrow SS$

   $S \rightarrow (S)$

   $S \rightarrow ()$

   And recognize the string of matching parenthesis. Generate and display the tokens and check for the errors in given input, if any. Use lex/flex or C/C++/Java for implementation of lexical analyzer.

Sample String	Sample output
()	S =>() Yes
(())	S=>(S) =>(()) Yes
((()))	S=>(S) =>((S)) =>((())) Yes
(())(	No

After completing the above, write the program for your own grammar.

5. 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	Yes
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.

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)

Important Requirement at the end of the course: Write your compiler for the Simple Object Oriented Programming Language using the guidelines given in the Decaf documents [Link is given in the course website]. Lex/Flex or YACC can be used for writing the compiler. If required you can use some simulators. However, the reasonable amount of work to be from your side.