Assignment of Compiler Design

1. Write a C program that read the following string:
   " Md. Tareq Zaman, Part-3, 2011"
   a) Count number of words, letters, digits and other characters.
   b) Separates letters, digits and others characters.

2. Write a program that read the following string:
   " Munmun is the student of Computer Science & Engineering".
   a) Count how many vowels and Consonants are there?
   b) Find out which vowels and consonants are existed in the above string?
   c) Divide the given string into two separate strings, where one string only contains the words started with vowel, and another contains the words started with consonant.

3. Write a program that abbreviates the following code:
   ***CSE-3141* as Computer Science & Engineering, 3rd year, 1st semester, Compiler Design, Theory.**

4. Write a program to build a lexical analyzer implementing the following regular expressions. It takes a text as input from a file (*e.g.,* input.txt) and display output in console mode:

Integer variable = (i-nI-N)(a-zA-Z0-9)*
ShortInt Number = (1-9)|(1-9)(0-9)|(1-9)(0-9)(0-9)|(1-9)(0-9)(0-9)(0-9)
LongInt Number = (1-9)(0-9)(0-9)(0-9)(0-9)+
Invalid Input or Undefined = Otherwise

5. Write a program to build a lexical analyzer implementing the following regular expressions. It takes a text as input from a file (*e.g.,* input.txt) and display output in console mode:

Float variable = (a-hA-Ho-zO-Z)(a-zA-Z0-9)*
Float Number = 0.(0-9)(0-9)|(1-9)(0-9)*.(0-9)(0-9)
Double Number = 0.(0-9)(0-9)(0-9)+|(1-9)(0-9)*.(0-9)(0-9)(0-9)+
Invalid Input or Undefined = Otherwise

6. Write a program to build a lexical analyzer implementing the following regular expressions. It takes a text as input from a file (*e.g.,* input.txt) and display output in console mode:

Character variable =ch_(a-zA-Z0-9)(a-zA-Z0-9)*
Binary variable = bn_(a-zA-Z0-9)(a-zA-Z0-9)*
Binary Number = 0(0|1)(0|1)*
Invalid Input or Undefined = Otherwise

7. Write a program to recognize C++
   i) Keyword ii) Identifier iii) Operator iv) Constant

8. Write a program which converts a word of C++ program to its equivalent token.

RESULT:

Input: 646.45
Output: Float
Input: do
Output: Keyword
Input: 554
Output: Integer
Input: abc
Output: Identifier
Input: +
Output: Arithmetic Operator

9. Write a program that will check an English sentence given in **present indefinite** form to justify whether it is syntactically valid or invalid according to the following **Chomsky Normal Form**:

S → SUB PRED
SUB → PN | P
PRED → V | V N
PN → Sagor | Selim | Salma | Nipu
P → he | she | I | we | you | they
N → book | cow | dog | home | grass | rice | mango
V → read | eat | take | run | write

10. Write a program to implement a shift reducing parsing.

11. Write a program to generate a syntax tree for the sentence a+b*c with the following grammar:

E → E+E|E-E|E*E|E/E|(E)|a|b|c

12. Write a program to build a lexical analyzer implementing the following regular expressions. It takes a text as input from a file (*e.g.,* input.txt) and display output in console mode:

E → E A E | (E) | ID
A → + | - | * | /
ID → any valid identifier | any valid integer

RESULT:

Input: Enter a string : 2+3*5
Output: VALID
Input: Enter a string : 2+*3*5
Output: INVALID

13. Write a program to generate FIRST and FOLLOW sets using a given CFG.

14. Write a program to generate a FOLLOW set and parsing table using the following LL(1) grammar and FIRST set:

Grammar	FIRST set
E→ TE'	{id, (}
E'→+TE' | ϵ	{+, ϵ }
T → FT'	{id, (}
T' →*FT' | ϵ	{*, ϵ }
F→ (E) | id	{id, (}

15. Write a program to generate a parse tree of predictive parser using the following parsing table:

	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ϵ	E'→ϵ
T	T→FT'			T→FT'		
T'		T'→ϵ	T'→*FT'		T'→ϵ	T'→ϵ
F	F→id			F→(E)		

16. Write a program that converts the C++ expression to an intermediate code of Post-fix notation form.

RESULT:

Input:

Enter infix expression : (A – B) * (D/E)

Output:

Postfix : AB – DE / *

17. Write a program that converts the C++ statement to an intermediate code of Post-fix notation form.

RESULT:

Input:

Enter infix statement : **if** *a* **then if** *c-d* **then** *a+c* **else** *a*c* **else** *a+b*

Output:

Postfix : acd - ac + ac * ? ab + ?