University of Asia Pacific

Department of Computer Science and Engineering

Mid-Semester Examination Fall-2020

Program: BSc in Computer Science and Engineering

Course Title: Compiler Design **Course No.:** CSE 429 **Credit:** 3.00

Time: 1.00 Hour. **Full Mark:** 60

Instruction(s): Answer any three questions including 1 and 2.

1. a. Let L ={x,y} **[3**
Suppose you have constructed the following language: **+ 6**
"The set of all strings consisting of x or y which contains **xyy** somewhere in the string." **+ 3 =12]**

i) Write the regular expression for this language
ii) Convert the above regular expression into NFA.
iii) Show the string **xyyxyy** is supported by the above NFA or not. Mention the intermediary states.

b. Which phase is responsible for generating tokens? What are the tokens and lexemes and patterns of the below code? **[8]**

```
void main()
{
    int a = 4, b = 5, c;
    scanf("%d",&c);
    if(a != b && c>=0)
    {
        while(a %2==0)
         a=b+c;
        printf("The value of A nd B are: %d %d", a, b);
     }
}
```

2. a. What are differences between Lexical and Syntax Analysis Phase? Mention at least 3 differences including one example. **[8]**

b. Write regular expression for the following language **[4+4 =8]**

i) Identifier where Identifier can start with a letter or underscore followed by any number of letter, number, underscore or dollar sign. Here ∑=[a-zA-Z0-9_$]
ii) Unsigned Exponential Numbers. Here, ∑={0-9}
[Note: You do not have to worry about the fractional numbers.]

c) Write some limitations of Regular expression over Context Free grammar. **[4]**

3. a. The following is a grammar for regular expressions over symbols a and b only. **[12]**

REXPR → REXPR+ RTERM | RTERM
RTERM→ RTERM RFACTOR | RFACTOR
RFACTOR → RFACTOR* | RPRIMARY
RPRIMARY → a | b

i) Does the above stated grammar has Left factoring? If yes, eliminate that.
ii) Does left factoring make the grammar suitable for top-down parsing? How?
iii) In addition to left factoring, eliminate left recursion from the original grammar.
iv) Is the resulting grammar suitable for top-down parsing?

b. What is a recursive descent parser? Consider the following CFG: **[8]**

P→ Qx | Sw
Q→ xxR |xxyx
R→yyy
S→ xxT
T→ yyz

For the input string "xxyyzw" how recursive descent parser works?

Or,

4. a. Consider the following two CFGs: **[12]**

i) S→ F | De | (S+F)
F→ a | DB
B→cD | cDef | cDa | ef | ε
D→ abc| e| ε

ii) Q → QED | q
E → e
D → NFA | d
N → DFA | n
F → f
A → a

If **(last two digits of your ID%2==0)** use grammar 1 otherwise, use grammar 2. For predictive parser, generate the first and follow functions from the above grammar.
[Note: If the grammar has any problem(left factoring/ left recursion) eliminate that first, then implement first and follow function]

b. Consider the CFG: **[8]**

E → +EE | *EE | -EE | x | y

Consider a string +*-xyxy. How would you derive it by using the grammar? Find out if the grammar is ambiguous or not. Give proper reasons behind your choice.

Best of luck ☺