Assignment 5
Groups of 3
Due Date: April 30, 2025 (EoD)

Write (hand or digital) or type neatly. The solution should be clearly visible and easily readable. Submit a PDF.
*Write **NAMES & IDs at the top** of your solution.*
*Submit a single PDF file as: **Assign05.pdf***

For each of the following grammars, determine the First and Follow sets after left-factoring and/or eliminating left-recursion first. Show whether LL(1) property holds for each of the grammar.
First Production has the start symbol on LHS. Uppercase letters are non-terminals, while lowercase letters are terminal.

1) S → *a*ABC | $\mathcal{E}$
   A → *a* | *bb*D
   B → *a* | $\mathcal{E}$
   C → *b* | $\mathcal{E}$
   D → *c* | $\mathcal{E}$

2) A → BC*c* | *e*DB
   B → $\mathcal{E}$ | *b*CD
   C → D*a*B | *ca*
   D → $\mathcal{E}$ | *d*D

3) S → (X | E] | F)
   X → E) | F]
   E → A
   F → A
   A → $\mathcal{E}$

4) S → Xb | Yd
   X → aX | $\mathcal{E}$
   Y → cY | $\mathcal{E}$

5) S → MNOPQ
   M → m | $\mathcal{E}$
   N → n | $\mathcal{E}$
   O → o | $\mathcal{E}$
   P → p | $\mathcal{E}$
   Q → q | $\mathcal{E}$

6) S → A
   A → *a*B | A*d*
   B → *b*
   C → *g*

7) S → AaAb | *BbBa*
   A → $\mathcal{E}$
   B → $\mathcal{E}$

Prove that the following grammars are ambiguous or unambiguous.

1) S → *a* | aAb | *abSb*
   A → *a*AA*b* | *b*S

2) S → AB | *aaa*B
   A → a | Aa
   B → *b*

3) S → *xy*X*x*X | *xy*X*yy* | *yy*
   X → *xx*X | *yy* | *xx*

Consider the following CFG. Create the complete item sets, DFA for canonical closure, and the parse table for **LR(0)** parser.

S → AA
A → aA | b

Consider the following CFG. Create the complete item sets, DFA for canonical closure, and the parse table for **LR(1)** parser.

S → aAd | bBd | aBe | bAe
A→g
B→g

C++ Grammar:

1. Consider the ***Partial C++ Grammar*** given below in EBNF notation.
2. Be aware of the notational meanings.
   a. <symbol> represents non-terminals, while without the < > is a terminal.
   b. * represent Kleene star, i.e., <ostream*> means zero or more occurrence. It is same as
      `<ostream> ::= <expression> << <ostream> | ε`
   c. ::= is same as →
   d. | is the same as or
   e. ( ) and { } are considered as brackets that are part of syntax

3. **There are several issues/mistakes/problems in the given CFG. Identify as many of these as you can, and correct them.**
4. **Add the following constructs to the CFG**
   a. **Switch Case**
   b. **Constant variable**
   c. **Classes //Keep it basic. Inheritance is not required for this lab.**
   d. **Try Catch**
   e. **Enforce precedence in <expression> when <arithmetic> is used.**
5. Any change in the current productions has to be RED in color. Any new production added has to be in BLUE color.

The following is also available on Lexue as a text file.

<program> ::= <fun-def*> int main(void) { <block> }

<block> ::= { <var-def*> <statement*> }

<fun-def> ::= void <var-name> (<parameters>) { <block> }
| <type> <var-name> (<parameters>) { <block> }

<var-def> ::= <type> <var-name> ;
| <type> <var-name> { <expression> } ;
| <type> <var-name> = <expression> ;

<parameters> ::= <parameter_list>
| ε

<parameter_list> ::= <parameter>
| <parameter_list> , <parameter>

<parameter> ::= <type> <var-name>

<statement> ::= <var-name> = <expression> ;
| std::cout << <ostream*> <expression>;
| std::cin >> <istream> ;

```
            | <conditional> | <loop> | <update>
            | return; | return <expression>;

<istream> ::= <var-name>
<ostream> ::= <expression> <<

<expression> ::= <expression> <operation-2> <expression>
          | <operation-1> <expn>
          | ( <expression> )
          | <literal>
          | <var-name>

<operation-2> ::= <comparison>
          | <logical>
          | <arithmetic>

<operation-1> ::= - | !

<arithmetic> ::= + | - | * | / | %

<logical> ::= && | ||

<comparison> ::= == | != | < | <= | > | >=

<literal> ::= string
        | integer
        | floating_point_number
                | escape_sequence
        | character
        | true | false

<var-name> ::= var_func_name


<update> ::= <var-name> <operation-2>= <expression>;
        | <var-name>++; | <var-name>--;
        | ++<var-name>; | --<var-name>;


<conditional> ::=
    if ( expression ) { block }
  | if ( expression ) { block } else { block }
  | if ( expression ) { block } else if ( expression ) { block }
  | if ( expression ) { block } else if ( expression ) { block } else { block }

<loop> ::= while ( expression ) { block }
       | do { block } while ( expression );
       | for ( statement ; expression ; statement ) { block }
```