Team members:

1. Adham Anas 22010601
2. Badr ElSayed 22010664

1. Problem (1)

a. Problem Statement:

i. Design and implement a robust parser capable of handling logical expressions. The parser should support a simplified syntax for propositional logic, including the parentheses for grouping and the following operations
   1. negation (~)
   2. conjunction (^)
   3. disjunction (v)
   4. implication (>)

b. Used Data Structures:

i. **Stack**:
   1. To store the expression and change it from infix to postfix
   2. Use it to evaluate the postfix expression and return the result

ii. **HashMap**:
   1. Used it store the value (true/false) of all variables

c. Sample Runs:

i.

```
enter
(a>b)^(c v ~d)
a =(true/false) t
b =(true/false) t
c =(true/false) t
d =(true/false) f
true
```

ii.

```
enter
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~r
r =(true/false) t
true
PS C:\Users\adham\Desktop\lab1Discrete>
```

iii.

```
enter
(a>b)^(c v ~d)
a =(true/false) t
b =(true/false) t
c =(true/false) t
d =(true/false) f
true
```

iv.

```
enter
(a>b)^(c ~d)
Wrong expression
```

d. Assumptions and Details:

i. You should put the expression only in the console without any quotation marks or any sentence before it.
ii. Variables should be letters (can't take the same symbol as an operator ['~', 'v', '^', '>']). The uppercase and lower case of same letter are different variables.
iii. If you write into the expression '1', it means true and if you write '0', it means false.
iv. Evaluation starts by asking you for the truth value of each variable, as seen in the Sample Runs. You can either enter 't' or 'true' and 'f' or 'false'.
v. The value of variables are taken in evaluateExpression() method

2. Problem (2)

a. Problem Statement:

i. Design and implement inference engine that can apply inference rules to logical expressions. The engine should take logical expressions as input, identify applicable inference rules, and generate the corresponding output based on the rules' logic. The supported inference rules are as follows:

1. Modus ponens
2. Modus tollens
3. Hypothetical syllogism
4. Disjunctive syllogism
5. Resolution

b. Used Data Structures:

i. **ArrayList**:
   1. Storing the rules and expressions entered into the Inference Engine.
   2. Used to store each variable/operator of the expression for ease of accessing.

ii. **HashMap**:
   1. Maps the variables of the original premise of the rule to the variables of the user entered expression.

c. Sample Runs

i.

```
Enter 2 Expressions
p > ~Q
p
(Modus Ponens)
~Q
```

ii.

```
Enter 2 Expressions
~k > l
~l
(Modus Tollens)
k
```

iii.

```
Enter 2 Expressions
a>    ~b
~b > ~c
(Hypothetical Syllogism)
a>~c
```

```
Enter 2 Expressions
a>    ~b
b > ~c
The input expression cannot be inferred
```

iv.

```
Enter 2 Expressions
b v a
~a
(Disjunctive Syllogism)
b
```

```
Enter 2 Expressions
~a v b
a
(Disjunctive Syllogism)
b
```

v.

```
Enter 2 Expressions
q v ~a
a v p
(Resolution)
qvp
```

vi.

```
Enter 2 Expressions
~a^b

(Conjuctive Simplification)
~a
```

```
// MadeUp3 rule for testing:
exp1.setRepresentation(representation:"a^b");
exp2.setRepresentation(representation:"");
expOut.setRepresentation(representation:"a");
InferenceRule madeUp3 = new SimpleInferenceRule(exp1, exp2, expOut, name:"Conjuctive Simplification");
Iengine.addRule(madeUp3);
```

d. Assumptions and Details:

i. The 2 expressions are entered into the console without quotation marks. Put first expression, press “Enter”, put second expression.

ii. Variables should be letters (can’t take the same symbol as an operator [‘~’, ‘v’, ‘^’, ‘>’]). The uppercase and lower case of same letter are different variables.

iii. You can create new rules dynamically by making an instance of the class InferenceRule for each rule, insert 3 expressions into the arguments (2 for premises and 1 for conclusion). Then add them to the InferenceEngine -The 5 rules required are already instantiated-. EX: Sample Run: vi

iv. The order of the entered premises shouldn’t matter.

v. IMPORTANT: Each part is in a separate folder/project and is run through “App.java”.