Phase 1: Lexical Analyzer Generator

Name	ID
Youssef Mohamed Alshaarawi	74
Yahia Mohamed Abdelaziz Ragab Elsaadawy	70
Walid Ahmed Zattout	69
Marwan Ashraf Gouda	60

Problem Statement:

The task in this phase of the assignment is to design and implement a lexical analyzer generator tool.

Part I: Input Processing

Overview:

In this part, the input is processed in order to identify the lexical rules upon which the NFA will be built. Regular definitions are substituted in the given regular expressions to produce usable regular expressions.

Used Data Structures:

It was deemed fit to use an object-oriented model, thus the task was distributed among multiple classes each with a certain task:

- InputProcessor: acts as a façade to the program.
- RegexGenerationStrategy: used to implement the strategy design pattern.
- RegexGenerator: uses RegexGenerationStrategy to convert regular definitions to regular expressions.
- RegexListGenerator: used to implement the factory design pattern.
- RegularDefinition/RegularExpression

Part II: Regex Conversion to NFA

Overview:

In this part, the regular expressions generated in Part I are converted to a single non-deterministic finite automaton describing the whole language.

Used Data Structures:

- Vector
- Map
- String
- Node Class

Used Algorithm:

Used Thompson's Construction algorithm to convert strings into a graph connected by edges which are either character traversals or an epsilon edge where each regex string corresponds to a path in the graph.

Part III: NFA Conversion to DFA

Overview:

In this part, the non-deterministic finite automaton generated in Part II is converted to a deterministic finite automaton.

Used Data Structures:

- Vector
- Map
- String
- Node Class
- DFA_Node Class

Used Algorithm:

Used Subset Construction algorithm to convert NFA to DFA using breadth-first search to traverse the NFA and construct the corresponding DFA.

Part IV: DFA Minimization

Overview:

In this part, the deterministic finite automaton generated in Part III is minimized to reduce the number of nodes in the graph.

Used Data Structures:

- Vector
- Map
- String
- DFA_Node Class

Used Algorithm:

Used an algorithm which applies Equivalence Theorem minimize DFA by grouping equivalent classes into a single class hence decreasing the number of nodes.

Part IV: Lexical Analyzer

Overview:

In this part, the deterministic finite automaton generated in Part III is used to analyze input text files producing a sequence of tokens.

Known Issues:

We are yet to use the minimized DFA generated in Part IV to analyze input text files. However, the unminimized DFA of Part III produces the correct output.
