Author: Earl Jasper V. Aquitania (Group)
Cristine Kaye Fiesta
Sheila Mae Peralta
Section: BSCS 2B
Instructor: Bernard Gonzales

MACHINE PROBLEM

INTRODUCTION

The program is about generating truth tables for logical expressions involving variables (like P, Q, R, and S) and operators such as AND, OR, NOT IMPLIES, and EQUIVALENCE. It processes input equations and validates them. It also translates variables into a binary matrix for truth values, and displays the corresponding truth table. The program also includes mechanisms to detect and handle invalid equations and can handle expressions involving parentheses for operator precedence.

HOW IT WORKS / ALGORITHM(FLOWCHART OR PSEUDO-CODE)

The following pseudocode outlines the main function responsible for processing logical equations read from a text file. This function initializes necessary variables, reads equations, translates them, calculates their dimensions, and evaluates them using truth tables.

```
BEGIN MAIN FUNCTION
  # Phase 1: Initialization
   Initialize variable 'variable_used' to 0
   Create an empty list 'equations' to store logical equations

   # Phase 2: Reading Equations from File
   OPEN the file 'equation.txt' for reading
   FOR each line in the file:
      STRIP leading/trailing whitespace from the line
      IF the line ends with a semicolon ';':
         REMOVE the semicolon from the end of the line
         ADD the resulting equation to the 'equations' list

   # Phase 3: Processing Each Equation
   FOR each 'string_equation' in the 'equations' list:
      RESET variables for this equation:
```

```
    Set 'row' and 'col' to 0
    Reset 'variable_used' to 0
    Initialize 'translated_string_equation' as an empty string
    Initialize empty lists: 'solve_value', 'string_priority', 'p', 'q', 'r', 's'

PRINT the current equation being processed

# Phase 4: Translate the equation string
CALL 'translate()' function to:
    - UPDATE 'variable_used' with the count of unique variables
    - RETURN the translated version of the equation string as 'translated_string_equation'

# Phase 5: Calculate Dimensions for Truth Table
CALL 'calculate_dimensions()' function to:
    - COMPUTE the number of rows and columns based on 'variable_used'

# Phase 6: Fill Matrix with Binary Combinations
CALL 'fill_matrix_with_binary_count()' function to:
    - GENERATE a matrix of binary values representing all possible truth values combinations

# Phase 7: Assign Truth Values to Variables
CALL 'assign_values()' function to:
    - ASSIGN truth values to variables 'p', 'q', 'r', 's' based on the binary matrix
    - STORE the truth values in a dictionary called 'propositions'

# Phase 8: Store Translated Equation Values
CALL 'store_values_inside_the_array()' function to:
    - STORE and PROCESS the translated equation string
    - RETURN 'integer_equation' and 'solve_value_for_not'

SET 'not_length' to the number of NOT operations in the equation

# Phase 9: Validate and Rewrite Equation (if necessary)
CALL 'is_equation_valid()' function to:
    - CHECK the validity of the equation
    - IF necessary, REWRITE 'integer_equation' and 'string_equation'

# Phase 10: Calculate the Result of the Complex Equation
CALL 'calculate_complex_equation()' function to:
    - EVALUATE the equation based on the truth table and variables
```

```
    - RETURN the result in 'solve_value' and update 'string_priority'

  # Phase 11: Display the Truth Table
  CALL 'display_truth_table()' function to:
    - DISPLAY the evaluated truth table, including variable values and results for the equation

  PRINT three newline characters to separate this equation's result from the next

END MAIN FUNCTION
```

DESCRIPTIONS OF VARIABLES USED

In this document only some of the key variables used in the truth table implementation are explained. These variables play a crucial role in constructing , processing, and evaluating logical equations, as well as generating the corresponding truth table. While this explanation does not cover every variable, it highlights the main components involved in evaluating logical expression.

`string_equation`

It represents the logical equation provided by the user as a string (e.g., "(P AND Q)"). This is used for processing and displaying the equation.

`integer_equation`

This is a numeric or structured representation of the logical equation that mirrors string_equation, but in a form more suitable for computation or validation.

`variable_used`

It tracks the variables (like P, Q, R, S) that are used in the logical equation. It also helps to determine the scope of the truth table (e.g., how many rows for different combinations of truth values).

`row and col`

These variables represent the number of rows and columns needed for the truth table. The number of rows corresponds to the number of truth value combinations for the variables, while columns represent the number of variables and the evaluated expressions.

matrix

The binary matrix where each row represents a different combination of truth values (true/false) for the variables involved in the equation.

p, q, r, s

These variables represent the truth values (True/False) for logical variables P, Q, R, and S in the truth table. They are assigned values based on the binary matrix.

propositions

It is a dictionary that stores the truth values for the variables (P, Q, R, S) in the truth table. This is used to evaluate the logical expression.

solve_value_for_not

This variable is used to store the intermediate results or flags for handling NOT operations in the logical equation.

rewritten_string_equation

This variable holds a modified version of the original logical equation in string format, after any parentheses or complex structures have been processed or simplified.

rewritten_integer_eqeuation

This serves a similar role to rewritten_string_equation, but in a structured or numeric form more suitable for evaluation by the program.

solve_value

This variable stores the final truth values of the logical expression for each row of the truth table. It represents the evaluation of the entire logical equation for different combinations of truth values for the variables.

string_priority

This variable is used to track the order of operations or the priority in which parts of the logical expression should be evaluated.

ERROR HANDLING

1. Input Validation

- **Invalid Equation Detection**: The program checks for common issues with the logical expressions:
    - It verifies if the equation is enclosed in parentheses when required and checks for mismatched parentheses.
    - It ensures that operators are correctly placed (e.g., there must be an operator after a closing parenthesis and no operator before an opening parenthesis).
    - It prevents the use of consecutive variables without an operator between them.
- **Error Messages**: When an invalid condition is detected, the program uses `os.system('cls')` to clear the screen and provides specific error messages indicating the nature of the error, such as:
    - **"Error: Invalid equation detected."**
    - **"Error: Invalid use of opening brackets."**
    - **"Error: No operator after closing bracket."**
    - **"Note: Must be a valid equation only! (e.g., p and (q or r))"**
- **Exit**: After displaying the error message, the program calls `sys.exit(1)` to terminate execution. This prevents any further processing of invalid equations.

2. Equations Handling

- **Handling Different Cases**: The program has specific checks for various forms of equations (like those enclosed in parentheses or simple three-variable cases). For example:
    - If the input is of the form `(var op var)`, it rewrites it correctly.
    - It checks for and handles special cases where parentheses or operators might be incorrectly placed.

3. Function-Level Error Handling

- **Return Values for Error Indication**: Functions like **`is_equation_valid`** return **`(0, 0)`** if no errors are found. This allows the program to continue processing only valid equations.

4. Feedback for User Input

- The program provides guidance on valid input formats, which helps users understand how to construct their logical expressions correctly.

USER'S MANUAL

1. **Requirements**
    a. Install Python version.
    b. A text file named **`equation.txt`** containing the logical equations to be processed.
2. **Installation**
    a. **Download the Program**: Ensure you have the **`truth_table.py`** script.
    b. **Create a Text File**: Create a file named **`equation.txt`** in the same directory as the script. Write your logical equations in this file.
3. **Input Format**
    a. Each logical equation should be written on a new line in the **`equation.txt`** file.
    b. Each equation should end with a semicolon (**`;`**).
    c. The equations can contain the following variables: **`P`**, **`Q`**, **`R`**, and **`S`**.
    d. The operators supported are:
        i. **AND**: ^
        ii. **OR**: v
        iii. **IMPLIES**: =>
        iv. **EQUIVALENT**: ⇔

v. **NOT**: **~** (This is a unary operator and should precede a variable).

4. **Running the Program**
    a. Open your command line or terminal.
    b. Navigate to the directory containing **`truth_table.py`**.
    c. The program will read the equations from **`equation.txt`**, validate them, and generate the truth tables.

Outputs

- The program will display a truth table for each equation in the **`equation.txt`** file.
- Each truth table includes columns for each variable and the results of the logical operations.
- If an equation is invalid, the program will display an error message indicating the nature of the error.