February 18, 2023

Program Menu

- Project Title
- Project Proponents
- Project Co-proponents
- Project Proponent's Department
- Background of the Study
- Objectives
- Materials and Methods
- Limitations
- Plan for Dissemination & Findings
- Appendices
- Demonstration
- References

Presented by Group 1

Propologi Calculator: A Truth Table Generator Application Software with History Feature

Project Title

Discrete Mathematics and its Applications

Project Proponents

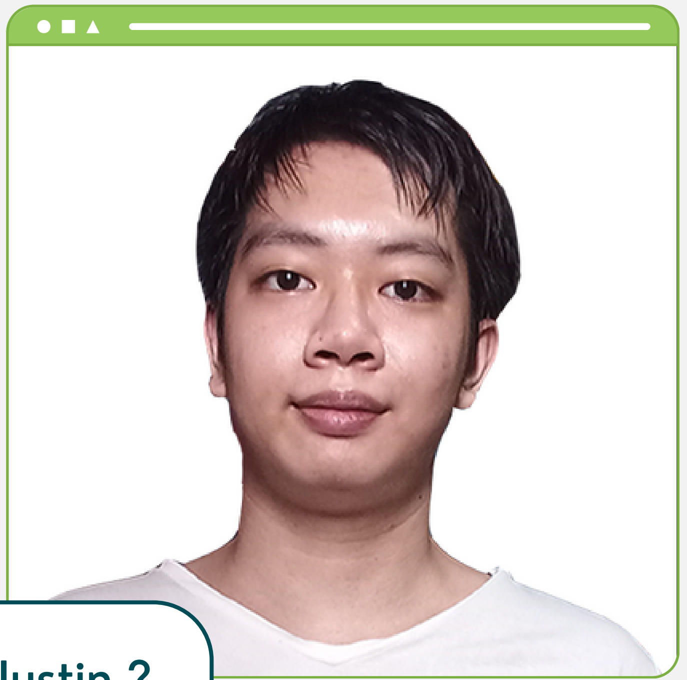
ALMEROL, Alvin L.
Leader

DE VILLA, Edrick L.
Business Analyst

OJEDA, Elliana B.
technical Writer

BOMBAIS, Alliah E.
System Analyst

MAGNAYE, Kyle R.
Developer



SALVADOR, Justin ?.
Project Designer

College of Computer and Information Sciences Computer Science

Bachelor of Science in Computer Science (BSCS)

Project Proponent's Department

Project Co-proponents

Ms. Angie Payne

MSIT, MCP

Discrete Structure's Professor

Background of the Study

Propositional logic studies the logical relationships and properties that result from methods of connecting or modifying statements in order to create more diverse and complex propositions, statements, or sentences. The study of propositional logic itself is complicated; therefore, making use of computer programs and the idea of portraying propositional logic through the concept of a truth table is an effective method for determining the values of various propositions. In addition, truth tables are an excellent approach to grasping the idea of how to analyze logical expressions and prove arguments. Due to its complexity, however, manual formulation and construction are considered time-consuming and tedious.

With that, various computer programs and applications, including the Propositional Logic Calculator, came into being. The functionality of the said software is essentially identical to that of a normal calculator application. However, the proponents of this project have seen an opportunity to improve the aforementioned software application. The program is designed to print outputs in the same manner as a manually constructed truth table, which is structured in rows and columns. However, based on the software capabilities in similar applications, in the event that during the printing of truth values, the user makes a mistake that causes the printed result to be cleared from the display, the user will be unable to look back on the truth values that have been solved. Hence, the proponent of this project came up with the idea of adding a feature that enables the users to look back at the previously solved truth tables, or, in summary, a history feature. The application provides users with a time-saving solution to the task of manually constructing truth tables for propositional expressions, a task that is both tedious and error-prone. Thus, this study was undertaken primarily with the purpose of assisting individuals such as students and anyone else who could personally benefit from the application's capabilities.

Objectives

- To create a Propositional Logic Calculator.
- To add a feature to a simple Propositional Logic Calculator, by allowing the users to access the previously solved truth tables.
- To improve the user experience of the mentioned application software.
- To obtain further knowledge about the implementation of discrete structures, specifically the concept of propositional logic through programming.

Materials and Methods

The project proposes a calculator which showcases (and computes for)

the concepts of propositional logic discussed throughout chapter 1

in this course's material Discrete Mathematics and Its Applications.

(Rosen, 2019)

This part is instructed to be skipped; therefore not Applicable

The proposed calculator focuses solely on working with the concepts of propositional logic, specifically in the generation of truth tables. These concepts include:

- Negation
- Conjunction
- Disjunction
- Conditional Connective
- Biconditional Logical Connective

The proposed calculator focuses solely on working with the concepts of propositional logic, specifically in the generation of truth tables. These concepts include:

- Finding the converse, inverse, and contrapositive of a propositional statement.
- Forming compound propositions.

The representation of a prepositional variable is denoted by the letters given within the calculator, which span from ‘p’ to ‘v’. These letters can be entered into the calculator to form a word that functions as one variable, but it is not intended to natively support full string cases.

The cases which can be entered into the calculator are virtually limitless; constrained only by the number of resources that Python can allocate for the calculator. The calculator can display a huge, tabulated result of indefinite length should the propositional case entered be complex. One can also extract the result from the calculator onto any text-editing program.

Limitations

Plan for Dissemination & Findings

As part of compliance with the Discrete Structures 2 course's finals requirements for all BSCS 2–4 students, the findings will be disseminated in the form of an application and research presentation. After scrutinization and approval by this course's instructor, Ms. Angie Payne, by giving a passing mark for this project proposal, the following steps for dissemination will be enacted:

1. **Uploading an open-source repository to GitHub.**

An open-source repository of Propologicalculator will be uploaded to GitHub. This allows developers and other researchers to download, use, and potentially contribute to the development of the application. The uploaded repo will include a readme file and any other dependencies needed for the application to work in a Windows OS environment – which is what was used for the development of this application.

2. **Used as a credential when presenting a Curriculum Vitae/Resume.**

As the program was developed for a major course requirement (Discrete Structures 2), all students involved in the development of this program may cite Propologicalulator as one of their qualifications and/or notable achievements. As it is uploaded to GitHub, one can easily inspect and assess the program's integrity and presentation at will.

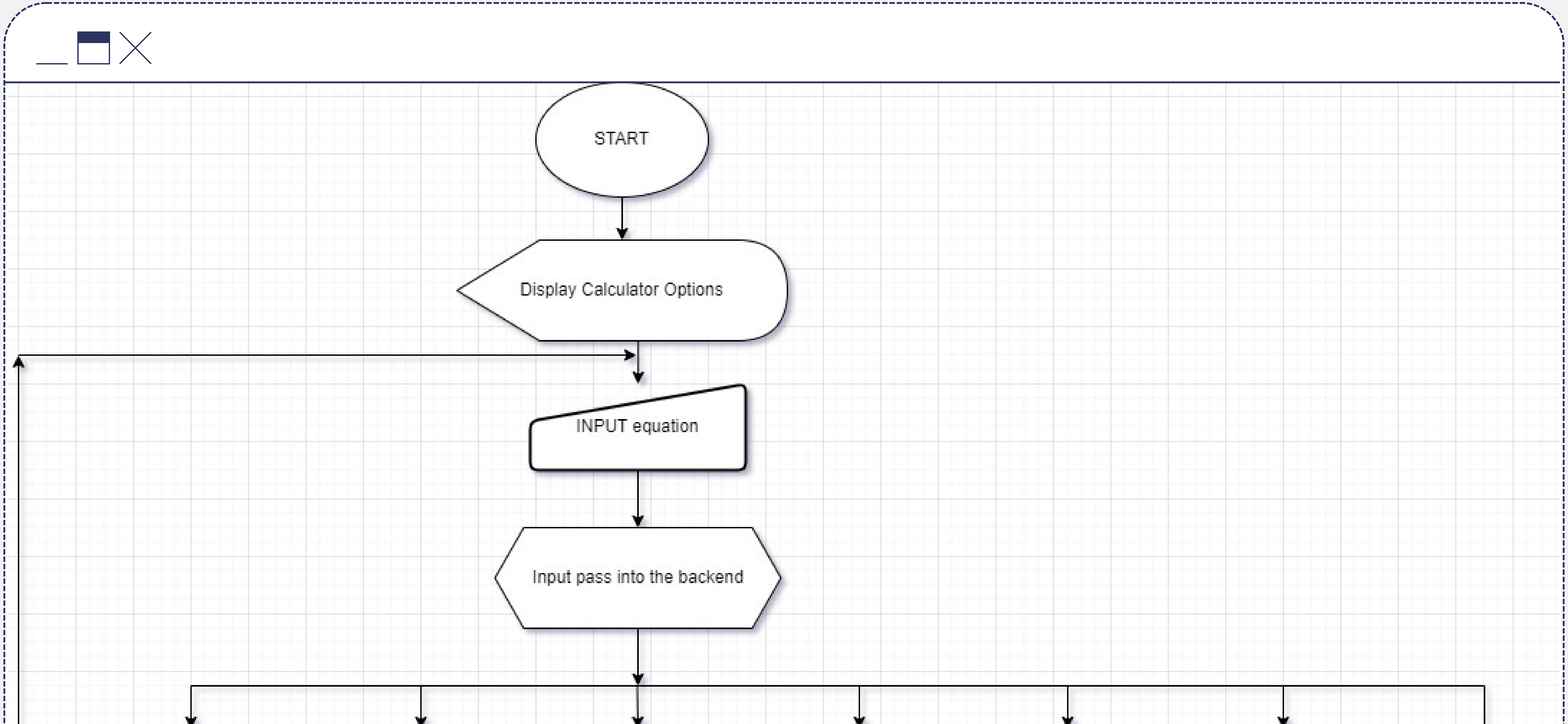
3. **Publication to a public journal hosting site.**

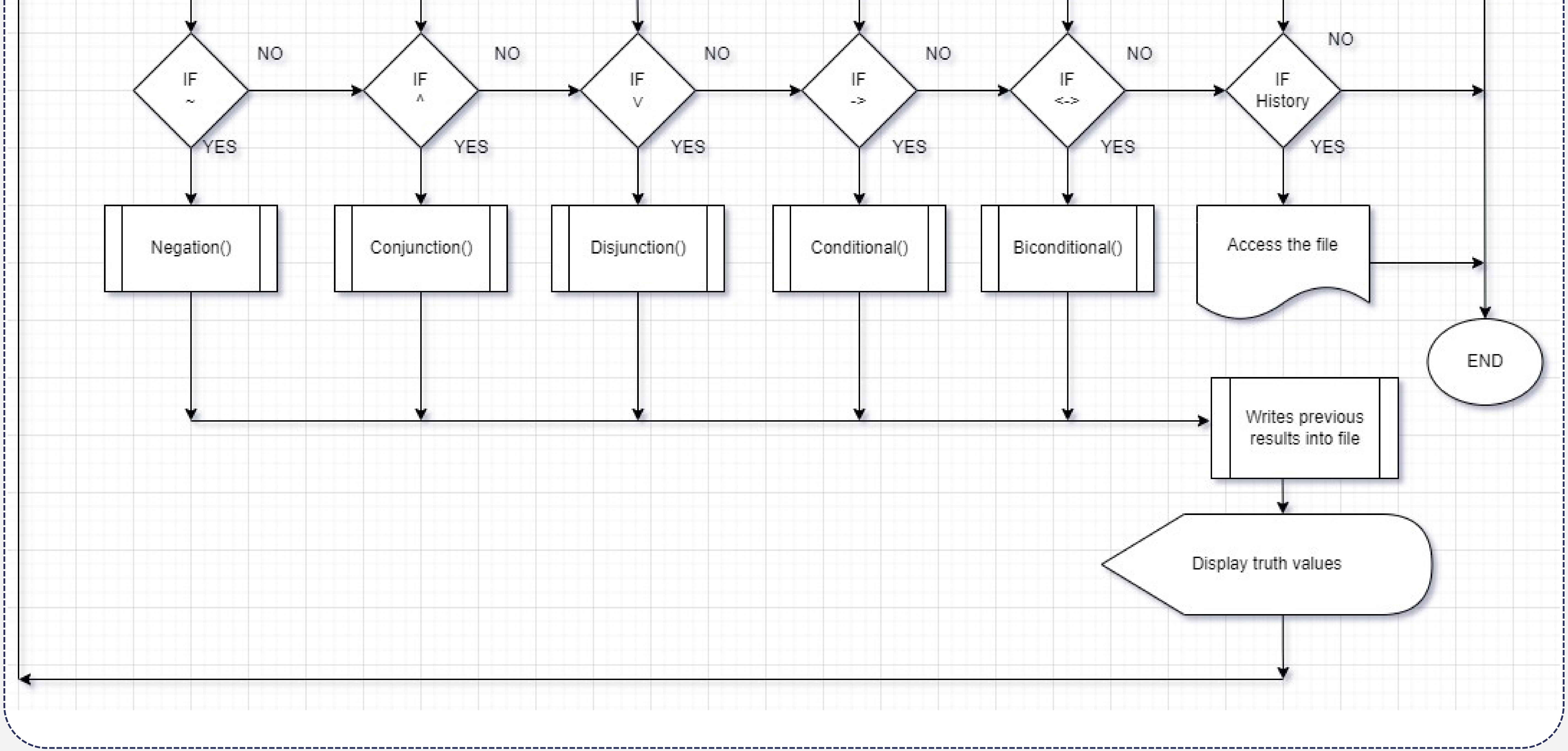
This project proposal could be uploaded to a public journal hosting site, such as Google Scholar or ResearchGate, which may include a document that details the algorithms used to calculate propositional logic equivalences. This makes the project accessible to other students or researchers bearing an identical study and allows them to use this proposal as a resource for their research. Furthermore, as the code is open source, it is possible for them to download and/or contribute to the expansion of this project.

4. **Collaboration with other students/used as future research.**

The program and proposal itself can be further disseminated by means of collaborating with other students should an idea involving the aspect of propositional logic. As the computer science course leans toward computation, it is likely that this project may remain relevant and be expanded upon by the members of this proposal later.

A. Flowchart


START
Display Calculator Options
INPUT equation
Input pass into the backend



B. Pseudocode

Appendices

B. Pseudocode

```
ALGORITHM Propositional Logic Calculator

 ALGORITHM Propositional Logic Calculator
FUNCTION isWellFormed(P)
    bracketLevel = 0
    FOR c IN P DO
        IF c == "(" THEN
            bracketLevel += 1
        END IF
        IF c == ")" THEN
            IF bracketLevel == 0 THEN
                RETURN False
            END IF
```

```
            END IF
            bracketLevel -= 1
        END IF
    END FOR
    RETURN bracketLevel == 0
END FUNCTION

FUNCTION parseNegation(P, truthValues)
    RETURN NOT parseProposition(P, truthValues)
END FUNCTION

FUNCTION parseConjunction(P, Q, truthValues)
    RETURN parseProposition(P, truthValues) AND parseProposition(Q, truthValues)
END FUNCTION

FUNCTION parseDisjunction(P, Q, truthValues)
    RETURN parseProposition(P, truthValues) OR parseProposition(Q, truthValues)
END FUNCTION

FUNCTION parseConditional(P, Q, truthValues)
    RETURN (NOT parseProposition(P, truthValues)) OR parseProposition(Q, truthValues)
END FUNCTION

FUNCTION parseBiconditional(P, Q, truthValues)
    RETURN parseProposition(P, truthValues) == parseProposition(Q, truthValues)
END FUNCTION

FUNCTION parseProposition(P, truthValues)
    P = P.replace(" ", "")
    IF NOT isWellFormed(P) THEN
        RETURN "Error"
    END IF
    WHILE P[0] == "(" AND P[-1] == ")" AND isWellFormed(P[1:len(P) - 1]) DO
        P = P[1:len(P) - 1]
```

```
WHILE P[0] == "(" AND P[-1] == ")" AND isWellFormed(P[1:len(P) - 1]) DO
    P = P[1:len(P) - 1]
END WHILE
IF len(P) == 1 THEN
    RETURN truthValues[P]
END IF
bracketLevel = 0
FOR i = len(P)-1 DOWN TO 0 DO
    IF P[i] == "(" THEN
        bracketLevel += 1
    END IF
    IF P[i] == ")" THEN
        bracketLevel -= 1
    END IF
    IF P[i] == "→" AND bracketLevel == 0 THEN
        RETURN parseConditional(P[0:i], P[i + 1:], truthValues)
    END IF
    IF P[i] == "↔" AND bracketLevel == 0 THEN
        RETURN parseBiconditional(P[0:i], P[i + 1:], truthValues)
    END IF
END FOR
bracketLevel = 0
FOR i = len(P)-1 DOWN TO 0 DO
    IF P[i] == "(" THEN
        bracketLevel += 1
    END IF
    IF P[i] == ")" THEN
        bracketLevel -= 1
    END IF
    IF P[i] == "∨" AND bracketLevel == 0 THEN
        RETURN parseDisjunction(P[0:i], P[i + 1:], truthValues)
    END IF
END FOR
bracketLevel = 0
```

```
    bracketLevel = 0
    FOR i = len(P)-1 DOWN TO 0 DO
        IF P[i] == "(" THEN
            bracketLevel += 1
        END IF
        IF P[i] == ")" THEN
            bracketLevel -= 1
        END IF
        IF P[i] == "∧" AND bracketLevel == 0 THEN
            RETURN parseConjunction(P[0:i], P[i + 1:], truthValues)
        END IF
    END FOR
    bracketLevel = 0
    FOR i = len(P)-1 DOWN TO 0 DO
        IF P[i] == "(" THEN
            bracketLevel += 1
        END IF
        IF P[i] == ")" THEN
            bracketLevel -= 1
        END IF
        IF P[i] == "¬" AND bracketLevel == 0 THEN
            RETURN parseNegation(P[i + 1:], truthValues)
        END IF
    END FOR
    RETURN "Error"
END FUNCTION

FUNCTION writeTruthTable(P)
    truthValues = {}

    FOR i FROM 0 TO length of P
        IF P[i] is a letter from "A" to "Z"
            SET truthValues[P[i]] = True
```

```
        SET truthValues[P[i]] = True
    END IF
END FOR

SET output to a new StringIO object
SET sys.stdout to output

FOR EACH statement IN keys of truthValues
    PRINT statement, " | "
END FOR
PRINT P

FOR EACH truthValue IN values of truthValues
    IF truthValue is True
        PRINT "T", " | "
    ELSE
        PRINT "F", " | "
    END IF
END FOR

IF parseProposition(P, truthValues) is True
    PRINT "T"
ELSE
    PRINT "F"
END IF

SET j to length of values of truthValues - 1

WHILE True in values of truthValues
    SET variable to the key at index j in keys of truthValues
    SET truthValues[variable] to not truthValues[variable]

    IF truthValues[variable] is False
```

```
        IF truthValues[variable] is False
            FOR EACH truthValue IN values of truthValues
                IF truthValue is True
                    PRINT "T", " | "
                ELSE
                    PRINT "F", " | "
                END IF
            END FOR

            IF parseProposition(P, truthValues) is True
                PRINT "T"
            ELSE
                PRINT "F"
            END IF
            SET j to length of values of truthValues - 1
        ELSE
            SET j to j - 1
        END IF
    END WHILE

    SET sys.stdout to the original stdout object
    RETURN the value of output as a string
END FUNCTION

FUNCTION Conjunction()
    Press the "V" key
    Release the "V" key
END FUNCTION

FUNCTION Conjuction()
    Press the "V" key
    Release the "V" key
END FUNCTION
```

```
FUNCTION Disjunction()
    Press the "∧" key
    Release the "∧" key
END FUNCTION

FUNCTION Conditional()
    Press the "→" key
    Release the "→" key
END FUNCTION

FUNCTION Biconditional()
    Press the "↔" key
    Release the "↔" key
END FUNCTION

FUNCTION Negation()
    Press the "¬" key
    Release the "¬" key
END FUNCTION

FUNCTION OpenP()
    Press the "(" key
    Release the "(" key
END FUNCTION

FUNCTION CloseP()
    Press the ")" key
    Release the ")" key
END FUNCTION

FUNCTION LetterP()
    Press the "P" key
```

```
    Release the "P" key
END FUNCTION

FUNCTION LetterQ()
    Press the "Q" key
    Release the "Q" key
END FUNCTION

FUNCTION LetterR()
    Press the "R" key
    Release the "R" key
END FUNCTION

FUNCTION LetterS()
    Press the "S" key
    Release the "S" key
END FUNCTION

FUNCTION LetterT()
    Press the "T" key
    Release the "T" key
END FUNCTION

FUNCTION LetterU()
    Press the "U" key
    Release the "U" key
END FUNCTION

FUNCTION LetterV()
    Press the "V" key
    Release the "V" key
END FUNCTION
```

```
FUNCTION Backspace()
    Press the Backspace key
    Release the Backspace key
END FUNCTION

CLASS ConsoleGUI
    FUNCTION __init__(master)
        // Initialize the Console GUI
        self.master = master
        self.frame = create a frame
        self.frame is packed to the top
        self.command_entry = create an entry widget for user input
        self.command_entry is packed to the bottom, filling the remaining space
        self.command_entry is set to execute the command when 'Enter' is pressed
        the focus is set to the command entry widget
        self.console_output = create a text widget to display output
        the widget is initially set to 'disabled'
        self.console_output is packed to the left, filling the remaining space

    FUNCTION execute_command()
        // Execute the user's command and display output
        get the command from the command_entry widget
        use the writeTruthTable() function to generate output based on the command
        set the console_output widget to 'normal' mode to enable writing to it
        add the output to the console_output widget
        set the console_output widget back to 'disabled' mode to prevent user input
        scroll the console_output widget to the end

    FUNCTION save_console_text()
        // Save console output to a file
        get the command from the command_entry widget
        delete the command from the command_entry widget
```

```
        delete the command from the command_entry widget
        use the writeTruthTable() function to generate output based on the command
        open a dialog box to choose a filename to save the output to
        if a filename is selected, create a new file and write the output to it

    FUNCTION clear_console()
        // Clear the console output and command entry widgets
        set the console_output widget to 'normal' mode to enable writing to it
        delete all text in the console_output widget
        set the console_output widget back to 'disabled' mode to prevent user input
        delete the command from the command_entry widget

    FUNCTION load_file(filename)
        // Load contents of file and return them
        open the file with the given filename
        read the contents of the file and return them

    FUNCTION load_file_handler()
        // Open a dialog box to choose a file to load and display contents in console output widget
        open a dialog box to choose a file to load
        if a file is selected, load its contents and display them in the console_output widget

    FUNCTION run()
        // Start the main loop for the program
        start the main loop for the program

Class PropologicalGUI:
    Function __init__(self, master):
        Set self.master to master
        Set the title of self.master to "Propological"
        Set the minimum and maximum size of self.master to 506 x 600 pixels
        Create an instance of the ConsoleGUI class and assign it to self.console

        Create 15 buttons with the following parameters:
```

Create 15 buttons with the following parameters:
    Text: "∨", "∧", "→", "↔", "¬", "(", ")", "P", "Q", "R", "S", "T", "U", "V", "⌫"
    Font: "Cambria" size 20
    Width: 6
    Foreground (text) color: white
    Background color: maroon
    Commands: Conjunction, Disjunction, Conditional, Biconditional, Negation,
                    OpenP, CloseP, LetterP, LetterQ, LetterR, LetterS, LetterT, LetterU, LetterV, Backspace
Place the buttons at the following (x, y) coordinates:
                    (20, 440), (140, 440), (260, 440), (380, 440), (20, 520), (140, 520), (260, 520), (20, 280),
                    (140, 280), (260, 280), (380, 280), (20, 360), (140, 360), (260, 360), (380, 360)

Create a "Done" button with the following parameters:
    Text: "Done"
    Font: "Cambria" size 20
    Width: 6
    Foreground (text) color: white
    Background color: maroon
    Command: execute_command method of self.console
Place the button at (380, 520)

Create a "Clear" button with the following parameters:
    Text: "Clear"
    Font: "Cambria" size 20
    Width: 6
    Foreground (text) color: white
    Background color: maroon
    Command: clear_console method of self.console
Place the button at (380, 130)

Create a "Save" button with the following parameters:
    Text: "Save"

```
            Font: "Cambria" size 20
            Width: 6
            Foreground (text) color: white
            Background color: maroon
            Command: save_console_text method of self.console
        Place the button at (380, 70)

        Create a "Load" button with the following parameters:
            Text: "Load"
            Font: "Cambria" size 20
            Width: 6
            Foreground (text) color: white
            Background color: maroon
            Command: load_file_handler method of self.console
        Place the button at (380, 10)

if __name__ == '__main__':
    Create a Tkinter root window and assign it to root
    Create an instance of the PropologicalGUI class and pass root as an argument, assign it to app
    Call the run method of self.console
END Propositional Logic Calculator
```

C. Logo

C. Logo

The PROPOLOGI-CALCULATOR Application has two logos, the main logo, and the mini logo. The first logo with the name of the application is the main logo. The logo has a minimalist look but without lacking details. It has this sleek look that adapts the classic calculator logos, yet, has its own vibe and colors with the propositional logic's logical symbols. This logo will welcome the users as its begins to run. The second logo on the right is the mini logo. This will be displayed at the top-left area of the application. Even though it is just the simpler version of the main logo, it carries the main details that will highlight the application's purpose.

PROPOLOGI-

CALCULATOR

Figure 1(left) & 2(right): PROPOLOGI-CALCULATOR Logos

```
bracketLevel = 0
for i in reversed(range(len(P))):
    if P[i] == "(":
        bracketLevel += 1
    if P[i] == ")":
        bracketLevel -= 1
    if P[i] == ">" and bracketLevel == 0
        return parseConditional(P[0:i],
    if P[i] == "=" and bracketLevel == 0
        return parseBiconditional(P[0:i]

bracketLevel = 0
for i in reversed(range(len(P))):
    if P[i] == "(":
        bracketLevel += 1
    if P[i] == ")":
        bracketLevel -= 1
    if P[i] == "|" and bracketLevel == 0
        return parseDisjunction(P[0:i],

bracketLevel = 0
for i in reversed(range(len(P))):
    if P[i] == "(":
        bracketLevel += 1
    if P[i] == ")":
        bracketLevel -= 1
    if P[i] == "&" and bracketLevel == 0
        return parseConjunction(P[0:i],

bracketLevel = 0
for i in reversed(range(len(P))):
    if P[i] == "(":
```

Application Development Process

It is also essential that we understand the exact procedures involved in app development.

In this segment, Our team will demonstrate the functions and structures of our application. In a nutshell, we will showcase the new and added features of our program..

Demonstration

References

Amos, D. (2022, July 18). Python GUI programming with Tkinter. Real Python. Retrieved December 16, 2022, from https://realpython.com/python-gui-tkinter/

Team, E. (2022, December 13). C program for a simple Calculator - C examples. Notesformsc. Retrieved December 16, 2022, from https://notesformsc.org/c-simple-calculator/

Rosen, K. H. (2019). Discrete mathematics and its applications. McGraw-Hill.

Chauhan Follow Web Developer & Content Writer at Student, O. (n.d.). Simple calculator flowchart. Share and Discover Knowledge on SlideShare. Retrieved December 16, 2022, from https://www.slideshare.net/OmiChauhan/simple-calculator-flowchart

ParthJadhav. (2022, September 27). Tkinter-designer/instructions.md at master · Parthjadhav/Tkinter-designer. GitHub. Retrieved December 16, 2022, from https://github.com/ParthJadhav/Tkinter-Designer/blob/master/docs/instructions.md#fformatting-1

The Collaborative Interface Design Tool. Figma. (n.d.). Retrieved December 16, 2022, from https://www.figma.com/

Klement, K. C. (n.d.). Propositional Logic. Internet encyclopedia of philosophy. Retrieved December 16, 2022, from https://iep.utm.edu/prop-log/

Leroy, K. (n.d.). Pseudocode Examples for Functions. OpenStax CNX. Retrieved December 16, 2022, from https://cnx.org/contents/MDgA8wfz@22.2:6Uvbhb3A@7/Pseudocode-Examples-for-Functions