Report of ACE CW1

1 PLogic

This Java program is about the calculation of propositional logic formula. Put-in the propositional logic formula , then the number of the total number of propositional variables involved in the formula, followed by truth value of each propositional variables, finally the program will output the truth value of the whole formula.

1.1 selection of data structures

- Array: Used to store unique propositional variables found in the input, e.g.: Character[] arr; hold the truth values of propositional variables, e.g.: Boolean[] truthValue
- List: Used to dynamically collect unique propositional variables found in the input formula, e.g.: List<Character> letter
- String: Used to represent propositional logic formulas and to manipulate substrings during calculations.
- Scanner: Used to read input from the user.

1.2 algorithm design

The program is designed to scan and collect the input propositional logic formula, the number of the total number of propositional variables involved in the formula, and each propositional variable's truth value. Then the program will calculate and output the truthvalue of the whole formula.

- Input scanning and checking: The 'Scanner' can read input by user and the method 'isLegal' can check if the formula is in right format.
- Check number of propositional variables: The program reads the total number of propositional variables 'num' and validates it.
- Read truth value: For each truth value input, it checks if it is a valid boolean value using the 'isBoolean' method. If the number is valid, it proceeds to read the truth values for each variable.
- Calculation:
  - Calculation prepare: Depending on the total number of propositional variables 'num', the program takes one of two calculation paths:
    If 'num' is greater than 0, it reads truth values and performs calculations using the calculate method, which handles cases where propositional variables are present.

If 'num' is 0, it skips reading truth values and directly performs calculations using the calculate method, which handles cases where there are no propositional variables.

- Calculation detail: The 'calculate' method plays a central role in evaluating the formulas by replacing subformulas with their truth values iteratively.
  It iteratively processes the formula until it evaluates to either "T" or "F." For formulas with propositional variables, it handles negation ' ~ ' , conjunction '&', disjunction '|', implication '->', and biconditional '<->' operators. The calculations involve updating the formula by replacing subformulas with their corresponding truth values.

- Output: The final result is printed to the console.
- Utility methods: Several utility methods are provided to check the legality of formulas 'isLegal and isLegalBrackets', validate boolean values 'isBoolean', count occurrences of characters in a string 'countChar', and find unique propositional variables 'findLetter'.
- Check error: The program includes error-checking mechanisms to handle various invalid inputs and formula structures. It can print error messages when wrong thing is detected.

1.3 algorithm correctness justification and efficiency analysis

The algorithm can run correctly after several different tests, as it successfully handles propositional logic formulas, validates inputs, and performs calculations.

```
PS C:\Users\86135\Desktop\ACE\CW 1\YuanWu_20411212> javac PLogic.java
PS C:\Users\86135\Desktop\ACE\CW 1\YuanWu_20411212> java PLogic
p
1
true
true
```

Figure 1: PLogic

The efficiency is reasonable, with a time complexity that depends on the length of the input formula and the number of propositional variables.

2 IdSum

This program is about the movement of students in a sequence. One student can move to anothers' right or left. One student can also change space with anothers'. Each student have an ID number. The program will execute the movement and return the sum of the ID numbers of the students at even positions in the resulting sequence.

2.1 selection of data structures

- Arrays: The student IDs are stored in an array 'student'.
- Lists: Used to store the parsed numbers obtained from user input, e.g.: 'List<Integer>'

- String: Used to represent user input for both the sequence of student IDs and the sequence of operations.
- Primitive data types: Integer variables 'int' are used to store various parameters like 'n', 'm', 'count', 'sum', and loop indices.

2.2 algorithm design

The program is designed to scan and part each number in the input, form a sequence of students, swap the order of the selected two students based on action number. Finally the program will calculate and output the sum of the ID numbers of the students at even positions in the resulting sequence.

- Read and check input values: Read a line of input to get the sequence of student IDs; check the validity of the input using the 'isLegal' method.
- Process the sequence of student IDs: The program can form a sequence of students according to the user's input, by: parsing the input string to extract student IDs into an array; initializing a loop to form the initial sequence of student IDs.
- Perform operations on the sequence:
  - Initialize a loop to perform operations based on user input by 'm' times.
  - Read a line of input to get the operation details.
  - Check the validity of the operation input using the 'isLegal1' method.
  - Extract the operation details and perform the specified operation on the 'student' array.
- Calculate the sum of IDs at even positions: The program can initialize a loop to calculate the sum of IDs at even positions in the modified sequence and finally print it as the output.

2.3 algorithm correctness justification and efficiency analysis

The algorithm can run correctly after several different tests, as it successfully change the position of students and output the right number of addition. The algorithm can also print error message while input is not in the correct format.

```
PS C:\Users\86135\Desktop\ACE\CW 1\YuanWu_20411212> javac IdSum.java
PS C:\Users\86135\Desktop\ACE\CW 1\YuanWu_20411212> java IdSum
6 3
1 2 3 4 5 6
1 5 2
2 3 4
3 1 6
10
```

Figure 2: IdSum

The efficiency is reasonable, with a time complexity of the program, which is influenced by the number of operations 'm' and the size of the student ID array 'n'. The loop iterating through the operations is $O(m)$, and the loop calculating the sum is $O(n)$. Therefore, the overall time complexity is $O(m+n)$.