**CPSC 331 Assignment#3**

Methodology***:***

The program starts with asking user input, after it askes user input. The user input will be passed into the method which takes the original logical expression entered by the user, and break it down so that all the independent variables from the original logical expression can be identified and placed into a list of strings. This will enable for the independent variables to be sorted in alphabetic order. This list of characters will be used later in the program to build the truth table. After I sorted the independent variables, I started to sort the logica subexpressions. The method which sorts logical expressions will use stack and dictionary. I used stack to store the index, I created a for loop for taking the original logical expression entered by the user, and break it down so that all the subexpressions from the original logical expression can be identified and placed into a list of strings. After I have all my subexpressions and independent variables, I started to deal with truth table. First, I check how many independent variables and how many subexpressions are in my List and Dictionary. Then I assigned them starting from the first column and first row of the 2-D array, and the column counter increment each time whenever I added my previous value into the array. In the end, I will have the first row of my 2-D array filled with all the labels. After finished the basic frame of the truth table, I started to compare the truth value for the first subexpression. In order to create an algorithm that separates the subexpression into premise, operand and conclusion. I created a stack, if it sees “(“, I push “(“ into the stack; if it sees “)”, I pop “)” out of stack, else if it sees “\*”, “-” or “+”, I checks if my stack is empty; if my stack is empty, then this sign will be the operand for sure; also, the index at this position will be where I need to cut my subexpression in half. Then I substring the subexpression between the operand into premise and conclusion. After I got my premise, operand and conclusion, now my life will be much easier! I allocated the premise and conclusion from the first row of my truth table since I have stored all the label in it in the previous step. I created a for loop to loop through all the label in the first row in order to check if there is any label equals to my premise and conclusion. Once my for loop found the index of the label in my truth table, I will be able to access the truth values from that column. I have my premise and conclusion truth values, I have my operand, ugh! I pass my three variables into the method that it compares two truth values with the operand and returns the result in boolean expression. When the operand is "or", the only case where the expression will be false is premise and conclusion are both false. When the operand is "negation", the only case where the expression will be true is premise is false. When the operand is "and", the only case where the expression will be true is premise and conclusion are both true. Right after my method returns a boolean expression, I will pass the truth value into the column where my subexpression is. After I have filled all the truth values into my entire subexpression column. The program starts to find the next subexpression and break it down into premise, operand and conclusion so on and so forth. After the truth table has been filled with the truth value. I print the table out by using a nested loop. Then the program ends.

Correctness of algorithm and termination:

**Part 1: independent Variable List**

{P}

Input:

A: a string representing a logical expression of length n specified by user input, a string is an array of characters with entries A[0],A[1],…,A[n-1] where an element at each index is type char.

{Q}

This method will take the original logical expression entered by the user, and break it down so that all the independent variables from the original logical expression can be identified and placed into a list of strings. This will enable for the independent variables to be sorted in alphabetic order. This list of characters will be used later in the program to build the truth table.

For(i:logExp.length()){p}

If(false)

End{q}

For(y:charArrayNoDuplicates.length()){p}

End{q}

{P} S {Q} since no matter what occurs a statement ill always happening concatenating to the string which is our Q and the for loop will eventually end when there are no more items in the string.

Therefore the loop will run (ith + 1)+(jth + 1) times since there are no loop variants that affect the loop whatsoever and the preconditions and postconditions are met since every {P} corresponds to a {Q}

**Part 2: subExpressions / expressions**

{P}

subExp:

A: a string representing a logical expression of length n specified by user input, a string is an array of characters with entries A[0],A[1],…,A[n-1] where an element at each index is type char.

{Q}

This method will take the original logical expression entered by the user, and break it down so that all the subexpressions from the original logical expression can be identified and placed into a list of strings. This process will occur via a for loop that iterates through the logical expression and match parenthesis in order to break it down into subexpressions. This Dictionary of subexpressions will be used later in the program to build the truth table.

For(b : subExp.length()){p}

If(true)

True{Q}

If(true)

True{Q}

{P} S {Q} since no matter what occurs a statement will always happening concatenating to the string which is our Q and the for loop will eventually end when there are no more items in the string.

Therefore the loop will run bth + 1 times since there are no loop variants that affect the loop whatsoever and the preconditions and postconditions are met since every {P} corresponds to a {Q}

**Part 3: Truth Table**

{P}

subs: it is a valid Dictionary

independentVar: it is a valid List

subExp: it is a valid string array

{Q}

This method will take the the subexpressions, independent variables. It calculates the truth values for all independent variables and later calculates all the truth values for the subexpressions. Return the 2-D array in the end.

For(a : independentVar.size()){p}

For(independentVar.size() : independentVar.size() + subExp.length){p}

For(c : independentVar.size()){p}

For(d : change){p}

For(e : row){p}

For(f : row){p}

For(g : subs.size()){p}

{P} S {Q} since no matter what occurs a statement will always happening concatenating to the string which is our Q and the for loop will eventually end when there are no more items in the string.

Therefore the loop will run (ath + 1) + (independentVar.size()th + 1) + ((cth+1) \* ((dth+1) \* ((eth + 1) + (fth + 1))) + (gth + 1) times since there are no loop variants that affect the loop whatsoever and the preconditions and postconditions are met since every {P} corresponds to a {Q}

**Part 4: Truth Value**

{P}

subs: it is a valid Dictionary

numberOfSubs: it is a valid integer

subExp: it is a valid string

table: it is a valid 2-D array

independentVar: it is a valid List

{Q}

This method will take the the subexpressions, independent variables. It calculates the truth values for all independent variables and later calculates all the truth values for the subexpressions. Return the 2-D array in the end.

For(a : subExp.length()){p}

If(true)

True{Q}

If(true)

True{Q}

If(true)

If(true)

True{Q}

True{Q}

If(true)

True{Q}

For(b : independentVar.size() + subs.size()){p}

If(false)

If(true)

If(true)

True{Q}

else(true)

If(true)

True{Q}

For(c : independentVar.size() + subs.size()){p}

{P} S {Q} since no matter what occurs a statement will always happening concatenating to the string which is our Q and the for loop will eventually end when there are no more items in the string.

Therefore the loop will run (ath + 1) + (bth + 1) + (cth + 1) times since there are no loop variants that affect the loop whatsoever and the preconditions and postconditions are met since every {P} corresponds to a {Q}

**Part 5: Truth**

{P}

firstSubs: it is a valid string

operands: it is a valid string

secondSubs: it is a valid string

{Q}

Output include:

This method will take first subexpression, operand and second subexpression as three parameters. It checks the different operands with the different true and false conditions from the first subexpression and second subexpression. It returns a boolean expression.

If(true)

True{Q}

If(true)

True{Q}

else(false)

If(true)

True{Q}

If(true)

True{Q}

else(false)

If(true)

True{Q}

If(true)

True{Q}

else(false)

{P} S {Q} since no matter what occurs a statement will always happening concatenating to the string which is our Q and the for loop will eventually end when there are no more items in the string.

Therefore the loop will run 1 time since there are no loop variants that affect the loop whatsoever and the preconditions and postconditions are met since every {P} corresponds to a {Q}

WORST CASE SCENARIOS:

**Part 1: independent Variable List**

Worst case is logExp.length() + charArrayNoDuplicates.length() for the for statements

**Part 2: subExpressions / expressions**

Worst case is subExp.length()

**Part 3: Truth Table**

Worst case is independentVar.size() + subExp.length + independentVar.size() \* (change \* (row + row)) + (subs.size() + 1)

**Part 4: Truth Value**

Worst case is subExp.length() + independentVar.size() + subs.size() + independentVar.size() + subs.size()

**Part 5: Truth**

Worst case is 1