CS2002 Practical 1 220025238 Tutor: rw86 14 Feb 2024

Overview

In this practical, I was tasked with developing an implementation of a program which prints truth tables for logical formulas given in reverse polish notation. I was then tasked with using my program to solve logic puzzles.

My solution achieved all parts of the specification.

Design and Implementation

Firstly, within ttable.c, I defined two constants which will dictate the maximum variables and formula lengths allowed. Within main, I ensured that there are 3 arguments present when running the program, including the name of the program. I then implemented a series of if statements to validate the arguments parsed upon execution of the program. This was to ensure a smooth, error free experience for the end user. If any of the arguments were invalid, an informative message was printed to the user informing them of that, and the program terminated. After checking the variable number and formula are of valid length, I parsed them as parameters onto a printTable() method.

Within printTable(), I used a bitwise left shift operator to calculate 2^n, and hence return the necessary number of rows I will need to print. I then created an integer array, with the number of variables as the size of the array. I used a for loop to print the appropriate letter variables in increasing order, as required by the specification, by adding the i variable to the character ‘a’, returning the correct letter variable for the variable number. I then used a series of printf statements to format the table as per specification requirements. Finally, I implemented a for loop which iterates over each row, calculating the values of each variable for a given row, depending on the binary representation of each row, using a bitwise right shift operator. I then printed the values for the current row, and calculated and printed the result value using the calculateFormula() method which takes in the formula, and the operands array as parameters.

Within calculateFormula(), I use a stack of size MAX_STRING_LENGTH constant. I then iterate over each formula character using a for loop in order to convert from reverse polish notation, to normal notation so I can compute the result of the logic formulas. For each character, I first check whether it is a lower case letter, then add the variable to the stack if so. I take the character that was read from the formula, and subtract it from the ‘a’ character, to return the appropriate operand index to top within the stack. I then use an else if statement to check whether the character is a constant (1 or 0) and add it to the stack, after converting it to an integer by subtracting the character ‘0’ from it. I finally use an else statement to check whether the formula character is an operator rather than an operand. Within else, I declare two operands, defining one of them by popping the operand from the stack immediately. I then check so that the current character is not a negation character, which will only need one operand rather than two. If it is not, I pop the stack yet again to define the second operand. If it is, I define it as 0. I then use a switch statement to go over each possible operator case. For each of the operators, I use the appropriate logical operators to compute the result and push it to the stack. Within the default statement, I print an error to inform the user that the formula is not following RPN (reverse polish notation) syntax. I then print the result of that specific operation underneath the operator on the truth table. Finally, I ensure that the stack is only contains one result, in order to validate the formula at the end of all of the operation calculations. If the stack does not contain one result, I inform the user that the formula is not the correct syntax and

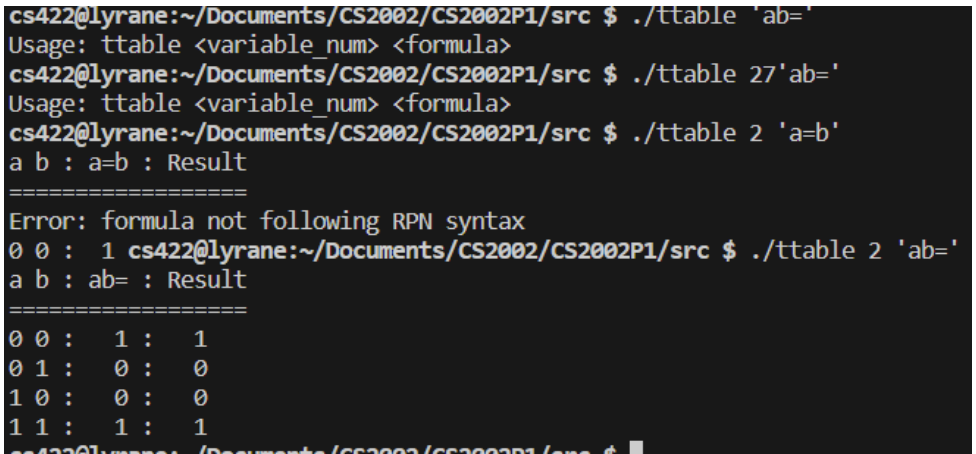
exit the program. If it is empty, I return the final top result and continue the for loop in the previous method until each row is printed correctly.

Testing

Note, all evidence is provided when stacscheck is ran and the following text is output:

StacsCheck
Testing CS2002 Logic
- Looking for submission in a directory called 'src': found in current directory
* BUILD TEST - build : pass
* COMPARISON TEST - prog-1var.out : pass
* COMPARISON TEST - prog-allops.out : pass
* COMPARISON TEST - prog-long.out : pass
* COMPARISON TEST - prog-spacing.out : pass
* COMPARISON TEST - prog-specexample.out : pass
* COMPARISON TEST - prog-spurious.out : pass
* COMPARISON TEST - prog-toolong.out : pass
8 out of 8 tests passed

What is being tested	Name of test method	Pre conditions	Expected outcome	Actual outcome
Checks program informs user if incorrect number of arguments parsed upon program execution.	main()	Incorrect number of arguments parsed.	Program informs user an incorrect number of arguments was parsed.	Program informs user an incorrect number of arguments was parsed.
Checks program informs user if invalid number of variables parsed upon program execution.	main()	Invalid number of variables parsed.	Program informs user an invalid number of variables was parsed.	Program informs user an invalid number of variables was parsed.
Checks program recognises invalid formulas.	calculateFormula()	Formula with incorrect syntax parsed as argument.	Program informs user the formula is not following the syntax required and halts.	Program informs user the formula is not following the syntax required and halts.



Using truth tables to solve Logical Problems

1) I firstly convert the following variant of De Morgan's Law to the appropriate syntax required by the program: ab|cd||-a-b-&c-d-&&=.
I then run it with 4 variables using the command: ./ttable 4 'ab|cd||-a-b-&c-d-&&='
This returns:

```
a b c d : ab|cd||-a-b-&c-d-&&= : Result
=======================================
0 0 0 0 :   0  001 1 11 1 1111 :   1
0 0 0 1 :   0  110 1 11 1 0001 :   1
0 0 1 0 :   0  110 1 11 0 1001 :   1
0 0 1 1 :   0  110 1 11 0 0001 :   1
0 1 0 0 :   1  010 1 00 1 1101 :   1
0 1 0 1 :   1  110 1 00 1 0001 :   1
0 1 1 0 :   1  110 1 00 0 1001 :   1
0 1 1 1 :   1  110 1 00 0 0001 :   1
1 0 0 0 :   1  010 0 10 1 1101 :   1
1 0 0 1 :   1  110 0 10 1 0001 :   1
1 0 1 0 :   1  110 0 10 0 1001 :   1
1 0 1 1 :   1  110 0 10 0 0001 :   1
1 1 0 0 :   1  010 0 00 1 1101 :   1
1 1 0 1 :   1  110 0 00 1 0001 :   1
1 1 1 0 :   1  110 0 00 0 1001 :   1
1 1 1 1 :   1  110 0 00 0 0001 :   1
```

This result returns true for each of the possible combinations of true/false values, meaning that the law is correct.

2) For this puzzle, I interpreted that there can only be one possible winner in a game of traditional heads and tails, therefore if Chris wins, Ian will lose and vice versa.
a = Chris wins
b = Ian wins
c = Coin lands on heads
a XOR b since only one possible winner.
c → a since if coin lands on head, Chris wins.
-c → -b since if coin lands on tails, Ian doesn't win.

The truth table returns the following:

```
a b c : ab#ca>&c-b->& : Result
==============================
0 0 0 :   0  10 1 110 :   0
0 0 1 :   0  00 0 110 :   0
0 1 0 :   1  11 1 000 :   0
0 1 1 :   1  00 0 010 :   0
```

1 0 0 : 1 11 1 111 : 1
1 0 1 : 1 11 0 111 : 1
1 1 0 : 0 10 1 000 : 0
1 1 1 : 0 10 0 010 : 0

Within both true results, a is true meaning that Chris won.

3) a=Ann; b=Barbara; c=Charles; d=Deborah; e=Eleanor.
d OR c
b XOR e
a → b
e ↔ d
c → (a AND d)

The truth table returns the following:

a b c d e : dc|be#&ab>&ed=&cad&>& : Result
==========
0 0 0 0 0 : 0 00 10 10 010 : 0
0 0 0 0 1 : 0 10 10 00 010 : 0
0 0 0 1 0 : 1 00 10 00 010 : 0
0 0 0 1 1 : 1 11 11 11 011 : 1
0 0 1 0 0 : 1 00 10 10 000 : 0
0 0 1 0 1 : 1 11 11 00 000 : 0
0 0 1 1 0 : 1 00 10 00 000 : 0
0 0 1 1 1 : 1 11 11 11 000 : 0
0 1 0 0 0 : 0 10 10 10 010 : 0
0 1 0 0 1 : 0 00 10 00 010 : 0
0 1 0 1 0 : 1 11 11 00 010 : 0
0 1 0 1 1 : 1 00 10 10 010 : 0
0 1 1 0 0 : 1 11 11 11 000 : 0
0 1 1 0 1 : 1 00 10 00 000 : 0
0 1 1 1 0 : 1 11 11 00 000 : 0
0 1 1 1 1 : 1 00 10 10 000 : 0
1 0 0 0 0 : 0 00 00 10 010 : 0
1 0 0 0 1 : 0 10 00 00 010 : 0
1 0 0 1 0 : 1 00 00 00 110 : 0
1 0 0 1 1 : 1 11 00 10 110 : 0
1 0 1 0 0 : 1 00 00 10 000 : 0
1 0 1 0 1 : 1 11 00 00 000 : 0
1 0 1 1 0 : 1 00 00 00 110 : 0
1 0 1 1 1 : 1 11 00 10 110 : 0
1 1 0 0 0 : 0 10 10 10 010 : 0
1 1 0 0 1 : 0 00 10 00 010 : 0
1 1 0 1 0 : 1 11 11 00 110 : 0
1 1 0 1 1 : 1 00 10 10 110 : 0
1 1 1 0 0 : 1 11 11 11 000 : 0
1 1 1 0 1 : 1 00 10 00 000 : 0
1 1 1 1 0 : 1 11 11 00 110 : 0
1 1 1 1 1 : 1 00 10 10 110 : 0

The only true result is when only Deborah and Eleanor attended, therefore they were the only people in attendance to the dinner according to the outputted truth table.

Conclusion

This practical allowed me to implement all of the theory from the lectures from both sides of the module in one project. This has allowed me to further experiment with C, implementing abstract data types such as stacks in a lower level language, all whilst exercising my understanding of truth tables and logic. Overall it was an interesting project, which I enjoyed designing and programming. If given more time, I would attempt to complete the final question from the second part of the coursework, and perhaps implement a more efficient way to check the formula syntax from the first part of the coursework, since my implementation checks the syntax after the formula is assumed to be correct, which sometimes may return erroneous results.