140012065 Ian 2nd February 2016

CS2002 Practical02

Overview:

For this practical we were asked to program a truth table in C. The truth table would then have to compute and print all rows of a truth table for a propositional formula. We had to make sure that the printing was formatted so that it would be easy for a human to read. We were also asked to prove De Morgan’s Laws and the Distributive Laws of Boolean algebra. The 3 step of the practical was to encode 3 puzzles into propositional logic.

Design:

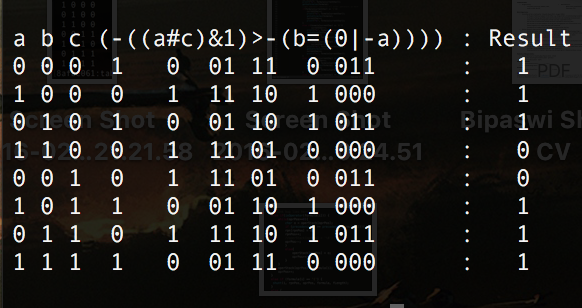
The designing of this implementation took a lot of thought as I found that there were several ways of doing it. The two main ways I realized that this practical could be done were through either a recursive descent parser or through a shunting algorithm. I ended up choosing the shunting algorithm which can be seen in my implementation. This is due to the fact that it seemed to be more straightforward to breakdown and evaluate the formula then by creating a parser. It also made sense to use the shunting algorithm as it meant that brackets could be ignored and hence making the evaluation far easier as all I would have had to do was pop from the stack and then push the evaluated numbers back. This also made the printing of the truth tables slightly easier to do as I could just add the results to an array that would then be printed out. In summary the way the program works is that a formula is first broken down into and rpn format through shunting. Once this is done it is then evaluated, this is where all the possibilities are passed into the variable place holders as well. Once a part of the formula is evaluated it is then stored in a results formula. This is then later printed out in conjunction with the formulas operators as can be seen in the screenshots making it easier to read.

I have put all my main code in the ttable.c file. This is mostly because at the time of writing it seemed natural to have the evaluation of the of the reverse polish notation in the same file as the shunting formula. However, in hindsight it might have been wiser to split the file up into ttable.c, shunt.c and eval.c. This would have been an improvement and made the code neater. However, as I have said at the time of writing it was easier to have everything in one file and not have to worry about how the files would fit together.

Testing:

The screenshots below show the testing the that was undertaken to make sure the program was robust and the true values of the truth tables was printed out. I also cross referenced my answers with a truth table generator online in order to check the correct values.

The below screenshot shows the examples being run.

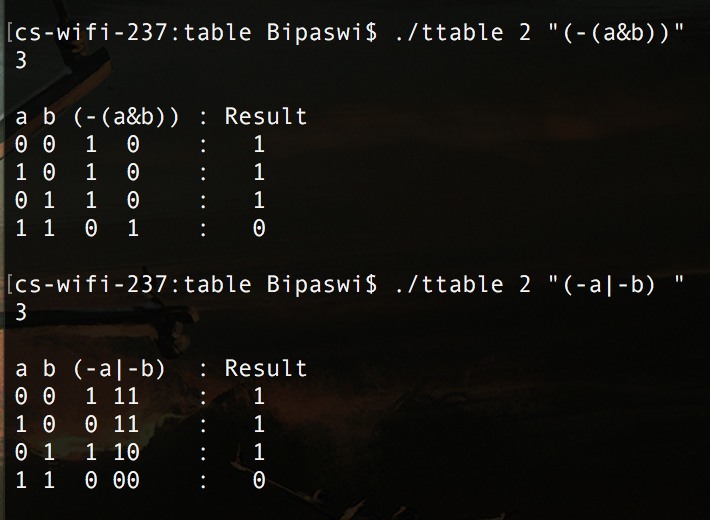


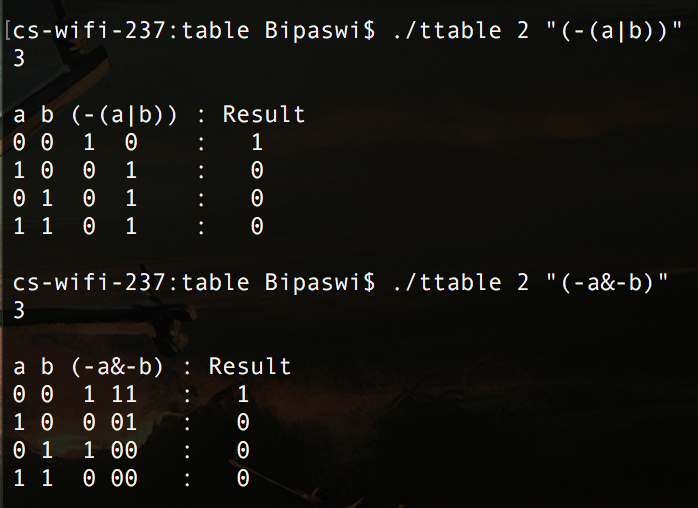
The below screenshot shows an invalid input being run in the program.



Step 2:

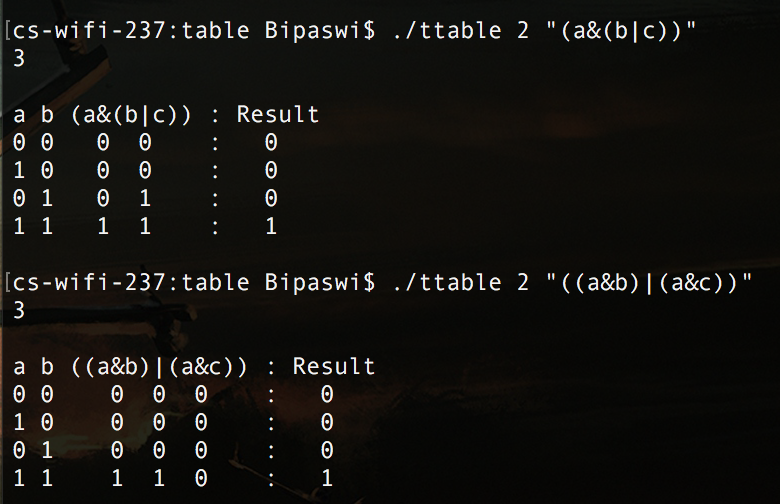
The two screen shots below show the proving of the de morgens laws:

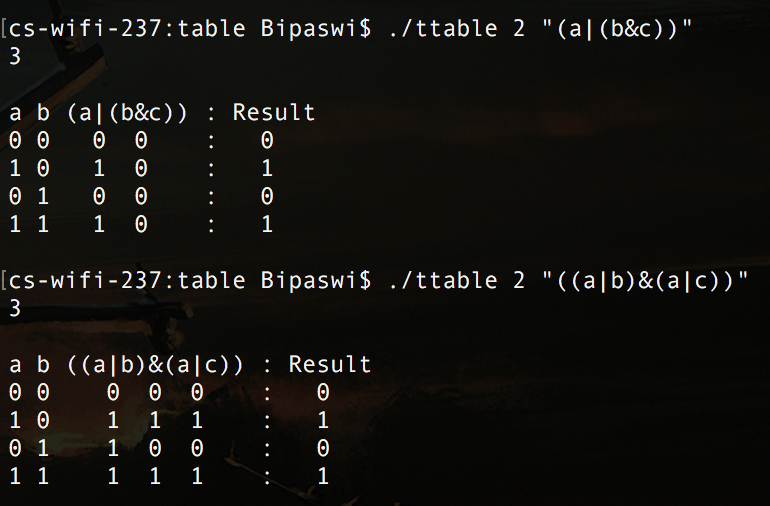
Too calculate the truth tables I calculated the left hand side with the right hand side of the equations. If the two equations and the same output in their truth tables then I assumed that they had proven de morgens law correct. 



The two screenshots below show the proving of the distribution law proof:

The way I have done this is by comparing the two formulas shown below. If the two formulas are the same then the the law will have been proven correct.





The following how the results for the way steps 3 A B and C were made.

Step 3:

A:

The way I work out puzzle A is in the following way:

A = Ian

B = Chris

C = Head

D = Tails

I then create the propositional formulas for each case. This is shown below:

If heads or tails : (c#d)

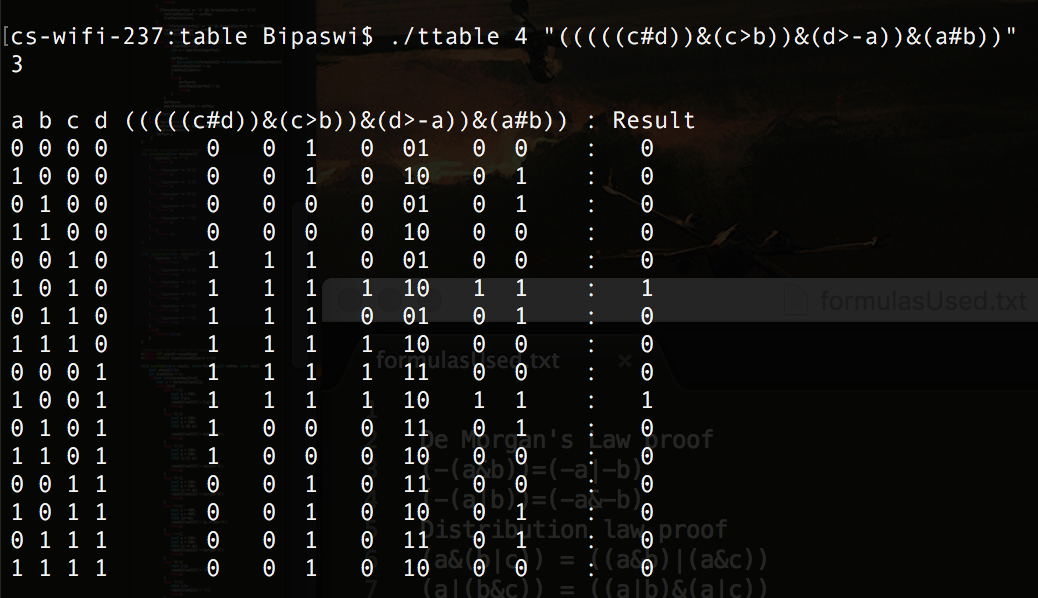
If coin is head then chris wins: (c>b)

If coin is tails then (d>-a)

Assume if only one wins (a#b)

Answer: Chris always wins.

I then put this in my program which then generates the following truth table. You can see that there is only one answer.



B:

The way in which I work it out for B is in the following way:

A = Chris Oldest

B = Steve Oldest

C = Ian Oldest

D = Chis Youngest

I then create the propositional formulas for each case. This is shown below:

Chris is oldest or Steve is oldest (a|b)

Both chris and ian cant be the oldest one (a#d)

Either chris is youngest or ian is oldest (c|d)

Either ian or steve is oldest not both (-b&c)

Either chris or ian is oldest but not both -(a&c)

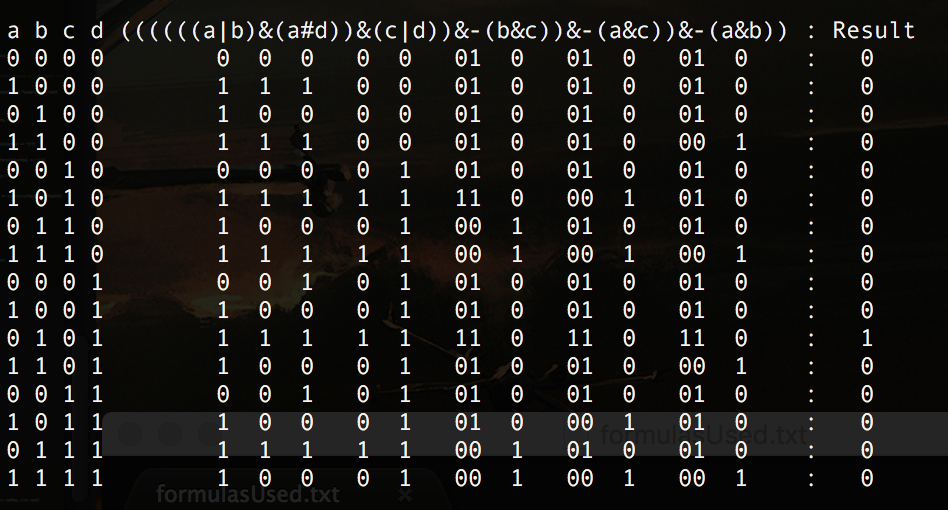
Either chris or steve is oldest but not both-(a&b)

Chris is Youngest

Ian is the middle age

Steve is Oldest

I then put this in my program which then generates the following truth table. You can see that there is only one answer.



C: The way in which I work out puzzle C is in the following way:

A = Ann

B = Barbara

C = Charles

D = Deborah

E = Eleanor

I then create the propositional formulas for each case. This is shown below:

If A attended, then so did B (a>b)

Either B or E but not both attended (b#e)

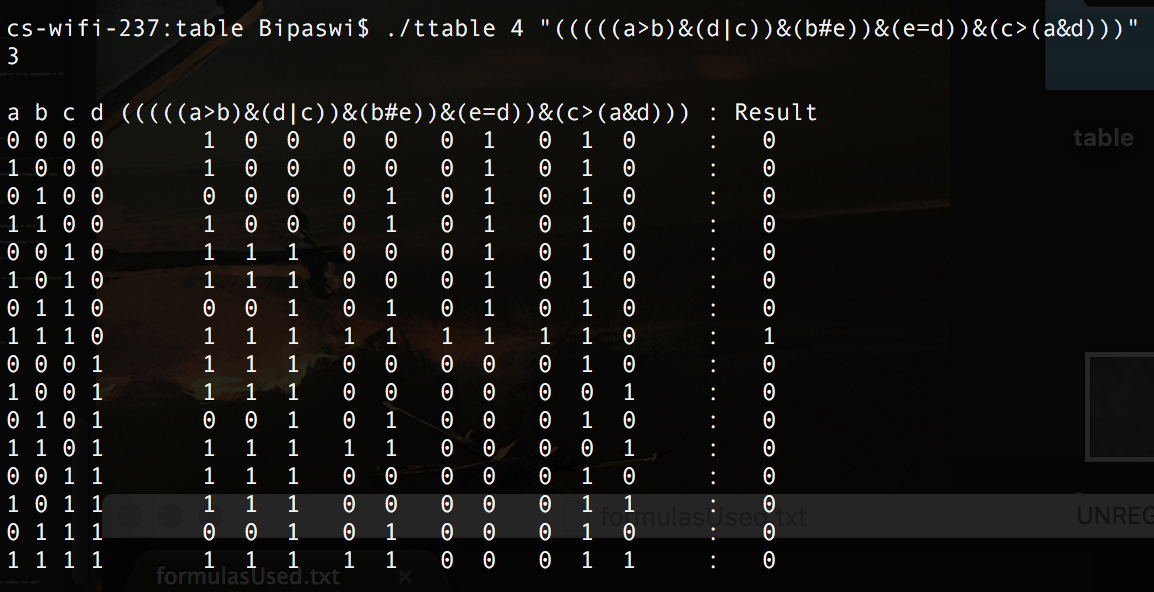
If C attended, then both A and D attended (c>(a&d)

Either D or C but not both attended dinner (d|c)

E attended if and only if D attended. (e=d)

Answer: Only D and E attended the party.

I then put this in my program which then generates the following truth table. You can see that there is only one answer.



Evaluation:

As I have said in my design aspect the splitting of files would have been useful and could also make the code a lot neater then it is at the moment. Other then the fact that the code was not split up into different files I feel that I have been able to successfully implement the specification that was given out. If given some more time, I would have liked to implement some of the extensions in step 4.

Conclusion:

I am now getting more comfortable with coding in C. Although it is vastly different from java. Doing this practical has been enjoyable however it has also had many challenges that have needed to be overcome. Such as figuring out the way in which to solve the coding part of the practical as well as the puzzles.