CMSC 2123 Discrete Structures

Project 1

Due: See the due date on D2L.

1. Basic Requirements

Count the number of true cases in the given compound propositional expressions, and determine whether each of them is a tautology, a contingency, or a contradiction:

1. $(p \vee q) \wedge (\neg p \wedge \neg q))$
2. $(p \leftrightarrow q) \rightarrow (\neg p \leftrightarrow \neg q)$
3. $(p \vee q) \wedge (\neg p \vee \mathrm{r}) \rightarrow (p \wedge r)$
4. $\big((p \rightarrow r) \rightarrow q\big) \leftrightarrow (p \rightarrow (q \rightarrow r))$

Your program must **compute** the truth values based on the operators and build a truth table for each expression: translate the symbols to the right representation and operators in C++.

The expressions can be hard-coded in your program, but the key is you need to use the right operators in C++ to get the right result that is consistent with the original logic expression. Regarding how to find the right operators, you can find some hints in the Output Specification section below.

2. Program Structure

2.1 Program Files

Project **1** consists of the file **p01.cpp.** Please save all your code in this file.

2.2 Output Specifications

Let $Math$ be the title for mathematical symbols and let $Disp$ be title for the symbols that are used for display by this project.

$Math$	$Disp$	$Math$	$Disp$	$Math$	$Disp$
$\wedge$	*	$\vee$	+	$\neg$	~
$\oplus$	^	$\rightarrow$	->	$\leftrightarrow$	<->
$\equiv$	===	*T*	*1*	*F*	*0*

The *Prog* symbols are not necessary operators in C++ programming. They are just commonly used in software for convenient display by any users. In this project, we are the "professional developers" of the software, so our language is different. You need to

choose the properly C++ operator to implement the required operation: please clearly distinguish *Math* symbols, *Disp* symbols and C++ operations in this project.

Please note:

(1). You don't need to list all the intermediate results for all the operators. But at least the columns for single variables and the final expression should be there. Please reference the "Sample Output" section: in this example, the first, the second and the fifth column are required, and the other columns are optional but can be helpful for debugging.

(2). After the calculation of the truth table, you need to recognize the true cases. Please also make a conclusion whether this expression is a contingency, a contradiction or a tautology, by analyzing the truth table (**not hard-coded conclusions**).

(3). Please use the right symbols defined above to **display** the expressions. You can use any methods to **compute** the truth values, e.g. use Boolean values or integers for the variables, use && and || for Boolean operators or use * and + for arithmetic operators depending on the value type for the variables. You can manually convert the given expressions to C++ expressions, and then hard code the C++ expressions in your program. Your program needs to further control all the input to these expressions.

(4). You **don't need to parse the expressions**, instead you can just represent them using the right notations/operators to replace the original symbols as discussed above (this is a manual translation in your program). After the manual conversion mention in item (3) above, your program can accept input variables' values and conduct the computation.

(5). Leave the converted expressions in your code directly. So you need to let the users to input the expressions or any other information. Once the jobs are done, terminate the program.

(6). Make sure your program can handle truth tables with multiple variables gracefully.

Pay attention: to display the expressions, you need to follow the symbols defined in the table above, but in your calculation, you can make you own design to choose the proper data types and operators. For example, in your calculation for $p \rightarrow q$, you can put $!p \,||\, q$ in the program to get the results (hint: $p \rightarrow q \equiv \neg p \vee q$), and display the title of this part as *p -> q*.

2.3 Sample Output

Below is an example regarding how to display the result for $(p \rightarrow q) \wedge (\neg p \vee q)$

In the first two columns, you need to generate different value combinations for the involved variables. Please make sure there are **no duplicates**.

The expressions are displayed using the symbols defined above in Output Specification section.

```
Check truth value for: (p->q)*(~p+q)

   p    q    p->q    ~p+q    (p->q)*(~p+q)

   0    0     1       1          1

   0    1     1       1          1

   1    0     0       0          0

   1    1     1       1          1

There are 3 true cases.
(p->q)*(~p+q) is a contingency.
```

2.4 Bonus task

1. Can you make the amount of code less than 150 lines? We need to practice clean code.

2. Can you parse the input expressions in your program? That is, the given expressions will be in string format in your program, and your program needs to map the symbols into the right operator to finish the calculation. If you have bonus task 2, you can create a new zip file named bonus2_zip in your submission, in addition to the basic version, so that we can easily verify whether bonus task 1 is achieved as well.

3. Submissions

Please submit a report along with your source code on D2L. This report should include your name, email, instructions to compile and execute your program (e.g., information about your operating system, command for compilation, etc.), and the result of a sample execution. All the files need to be zipped as **p01_group*.zip**, where * means your group number, e.g., the submission from group 3 should be named as p01_group3.zip.

If you have bonus task 2, you can create a new zip file named p01_group*_b2.zip in your submission, in addition to the basic version, so that we can easily verify whether bonus task 1 is achieved as well.

This is a team work, you can have a partner and register to the same group on D2L. Please check D2L course homepage -> “Communication” -> “Groups” -> “Project Groups”, and enroll yourself in a group.

4. Evaluation

This project will be evaluated according to the correctness of the various tasks specified above, and secondarily according to the quality of your code. The rubric is as follows:

Categories	Weights
`Data type design`	15%
`Operator design`	15%
`Expression translation`	15%
`Program framework`	15%
`Display`	15%
`Correctness`	15%
`Report`	10%:
`Bonus task 1`	Extra 10%
`Bonus task 2`	Extra 40%

Notes:

1. To be considered on time, the program must be turned in before the due date.
2. Programs must reflect your knowledge and work, not others.
3. No points, zero (0), will be given to any program containing compilation errors.