Report

- G1 enforces the intended precedence.

```
Formula   ::= Formula '<->' Formula | ImpTerm1
ImpTerm1 ::= ImpTerm1 '->' ImpTerm1 | ImpTerm2
ImpTerm2 ::= ImpTerm2 '\/' ImpTerm2 | ImpTerm3
ImpTerm3 ::= ImpTerm3 '/\' ImpTerm3 | ImpTerm4
ImpTerm4 ::=  '!' Formula | Factor
Factor     ::= '(' Formula ')' | ‘T’ | ‘F’ | Ident
```

- G2 enforces right-associativity of all binary operators along with precedence.

```
Formula   ::= ImpTerm1 '<->' Formula | ImpTerm1
ImpTerm1 ::= ImpTerm2 '->' ImpTerm1 | ImpTerm2
ImpTerm2 ::= ImpTerm3 '\/' ImpTerm2 | ImpTerm3
ImpTerm3 ::= ImpTerm4 '/\' ImpTerm3 | ImpTerm4
ImpTerm4 ::= '!' Formula | Factor
Factor     ::= '(' Formula ')' | ‘T’ | ‘F’ | Ident
```

Design Choices:

The program is divided into three files: **Parser**, **ParserHelper** and **P2_Jashanraj_Gosain.hs**

The program is divided into these files based on the responsibility (duty) of each of the function. Parser contains definition of **Parser** newtype, all the necessary function (parse in this case) and instances. There is a **parserHelper** file which define all the **parsing primitives**. **P2_Jashanraj_Gosain.hs** contain the **main module** which parse the given props using the parser defined in the file.

Program was divided to simplify the **complexity** of each individual file and reduce the duties carried by each individual module. On higher level, the duties were divided based on **parser definition, parser utilities and the main module**. **Parser definition** is a separate object/Unit of its on, so it was defined in a separate module. **Helper functions** for parser are the utilities of Parser that’s why it was assigned separately. This **reduces the effort** needed to maintain the program. These files don’t need to be dependent on each other. Parser definition can be changes **independent of other two** and same goes for the other two. **Additional functions can be added** in utilities and these utilities can be extended to other definitions of parsers. Similarly, P2_Jashanraj_Gosain can change module to change the definition of parser. This **structure reduces the interlinking of these three duties**, **however** these are **not completely independent** of each other.

Parser:

newType **Parser and Parser instances of Functor, Applicative, Monad and Alternative** are stored in a file this module. It uses imports from **control.Applicative, Data.Char**. This file exports **Control.Applicative, Data.Char** module along with including **Parser type, instances of Parser and parse function**.

ParserHelper:

This module contains all the **helper functions** for parser used in P2_Jashanraj_Gosain.hs file. This module exports every function **including imported 'Parser' module**. This stored functions like **sat, digit, lower, char, string, token, alphanum and others**.

P2_Jashanraj_Gosain.hs:
This file contains the **main module, var, constant, formula and helper functions for each**.

Constant :: Parser Prop

Constant is a Parser Prop value. It applies token function on **checkConstant** and gives its return value. **Token function** is applied to remove extra spaces from the starting of string.

CheckConstant :: Parser Prop

checkConstant is a Parser Prop value. It gives an parser which checks for "T" or "F" in the string and gives Parser with **(Const True) or (Const False)** respectively [a parser of Prop variable].

Var :: Parser Prop

Var is a Parser Prop value. It gives token function return on parameter **checkVar**. Token function is applied to remove extra spaces from the starting of string.

CheckVar:: Parser Prop

checkVar is a Parser Prop value. It gives an parser which **checks for a lower case char value** in the string and if it finds lowercase value, it applies **many on alphanum** that is it keeps finding digit or char until it fails. It gives Parser with **(var stringValue)** [a parser of Prop variable].

Formula :: Parser Prop

Formula applies token to **checkFormula** in order to remove any form of spaces in between the string values.

Like formula, there are functions **impTerm1, impTerm2, impTerm3, impTerm4 and factor** based on the grammar of G2. All these functions apply token to respective **checkfunction** for them. This is done for the same reason to remove whitespaces.

Signature types of **impTerm1, impTerm2, impTerm3, impTerm4 and factor** is also same as formula that is `**:: Parser Prop**`.

To create a parser **following right associativity and precedence** as depicted in **G2** representation.

Formula applies a do block to check for a pattern that matches "**impTerm1 <-> formula**". If that fails, it goes to **impTerm1**. If it matches the former, it will return a parser (**Iff impTerm1 formula**) which will be computed **recursively**.

ImpTerm1 applies a do block to check for a pattern matching "**impTerm2 -> impterm1**". If that fails, it goes to **impTerm2**. If it matches the former, it will return a parser (**Imply impTerm1 formula**) which will be computed **recursively**.

ImpTerm2 applies a do block to check for a pattern matching "**impTerm3 \/ impterm2**". If that fails, it goes to **impTerm3**. If it matches the former, it will return a parser (**Or impTerm1 formula**) which will be computed **recursively**.

ImpTerm3 applies a do block to check for a pattern matching "**impTerm4 /\ impterm3**". If that fails, it goes to **impTerm4**. If it matches the former, it will return a parser (**And impTerm1 formula**) which will be computed **recursively**.

ImpTerm4 applies a do block to check for a pattern matching "**! Formula**". If that fails, it goes to **factor**.

If it matches the former, it will return a parser (**Not formula**) which will be computed **recursively**.

Factor applies a do block to check for three patterns: **"(formula)", any constant value or any variable**.

This pattern **ensures** that the **parser is right associative and follows the precedence** : : (), !, /\, \/, ->, <-> (from high to low).

ParseFormula function has **type signature :: String -> String**. This function takes in **string value** and parse it according to the above defined parser (following the given patterns, symbols, right-associated and the given precedence). This function has a case base parse formula value. **Parse function** is used from the ParserHelper file. This function applies the parser formula on the value and returns a **result in the form of [(a, string)]**. In this case, the output type is [(Prop, string)]. For **error case, it returns []** or **unfinished parsing that is string value in the pair inside array is not empty**. **Otherwise**, it will be considered a **success case**. This parser wouldn't take into account multiple parsing value as the defined grammar in G2 is unambiguous. In other words, there will be no more than one pair returned as array elements by the defined parser.

Main :: IO()

The last section consists of the **main module** of the file. This module takes in **FileName** as **argument** from the user and **reads** the **indicated file** data assuming **each line** is a **prop**. It then **parse** each **prop using formula** function and **prints** the **result** for each of the props.