Project Assignment Report

All the data types and functions are included in a single file. The file is divided into parts for each function. The structure was small enough to be designed in a single file and it's easy to follow.

Design Choices:

The program is contained in a single file, namely **P4_Jashanraj_Gosain.hs**

Functions and definitions are unified (in a single file structure) based on the duty/functionality of the program. All the functions have related functionality, therefore defined in a single file. Types, functions, and type checking are interdependent and change in any function will advice updates in other functions and types. It is viable to maintain and update the program without having to change other files when placed in a single file. As each function will probably need some alteration if any of the other function is changed, it is better to keep the dependent functions in a single file structure. It will reduce the need to alter outside environments/files every time any function/type is changed which prevents foreign environment affecting the local environment and provide a robust system.

The file is divided into four parts: Imports, type definitions, functions, Main module.

P4_jashanraj_gosain.hs **imports module** Data.Map.Strict, Data.Set, Control.Monad.State.Lazy, Prelude, System.Environment and System.IO.

Type and data declaration part defines VarId, Type, Expr, Constraint, Env type, InferState, RelabelState and Substitution.

- VarId is string type.
- data Type has five data constructors 'TInt', 'TBool', 'TError', 'TVar Int', 'TArr Type Type' and it derives Eq, Ord, Read and Show classes.
- Expr has data constructors for CInt, CBool, Var, Plus, Minus, Equal, ITE, Abs, App, LetIn and it derives Eq, Ord, Read and Show classes.
- data Constraint has two data constructors 'CEq Type Type' and 'CError' and it derives Eq, Ord, Read and show classes.
- Type ConstraintSet is Set.Set Constraint
- Type 'InferState a' is State Int a
- Env is a Map which takes VarId as key and Type value.

- Type Substitution is Map.Map Type Type

Functions:

relabel is a function with signature:: Type -> Type. It was incorporated from the appendix the project as per guidelines
getFreshTVar is a function with signature:: InferState Type. It gets a fresh TVar value which is not used before.

infer is a function with signature:: Env -> Expr -> InferState (Type, ConstraintSet). It takes in **two** parameters of type 'Env' and 'Expr'respectively. It implements the CT-Int, CT-Bool, CT-Var, CT-Plus, CT-minus, CT-Equal, CT-ITE, CT-Abs, CT-App, CT-LetIn. It returns the result in InferState (Type,ConstrainSet).

inferExpr is a function with signature:: Expr -> (Type, ConstraintSet). It takes in an Expr and applies the infer function to that Expr with an empty environmemt. The result is returned in the form of (Type,ConstraintSet). It evaluates the type and constraints value of the expression.

toCstrList has a type signature:: ConstraintSet -> ConstraintList. It converts the constraints from Set form to list form using Set.toList function.

applySub has a type signature:: Substitution -> Type -> Type. It takes in a Substitution value and Type value. It applies substitution on the type and returns the obtained result after applying substitution to the given Type value.

applySubToCstrList has a type signature:: Substitution -> ConstraintList -> ConstraintList. It takes in Substitution and constraintList. It applies the substitution to each constraint in the list and returns the resulting constraintList after substitution.

composeSub has a type signature:: Substitution -> Substitution -> Substitution. It takes in two substitution values and uses fmap to apply sub S1 to S2 substituion and then union it with S1 substitution.

TVars has a type signature:: Type -> Set.Set Type. It finds the TVar values in the type and returns a set of TVar values in the Type.

unify has a type signature:: ConstraintList -> Maybe Substitution. It takes in a constraintList and find most general substitution for the constraintList using unification. Its return value is Maybe Substitution cause it returns nothing if the constraints have CError or some other error.

typing has a type signature:: Expr -> Maybe Type. It takes in an expression and applies inferExpr on that expression. It applies unify to the constraints returned by inferExpr and return the result of applysub on unified constraints and the type.

typeInfer has a type signature:: Expr -> String. It takes in a expression and applies typing on the expression. If the return value of typing is Nothing, it returns “Type Error”, otherwise returns “show (relabel v)” where v is the value wrapped inside (Just v).

The last section consists of the **main module** of the file. This module takes in FileName as arguments from the user and reads the indicated file data assuming each line is a FUN language expression. It then converts string lines to Expr object using read function and prints the result of mapping typeInfer function over the list of Expr objects returned by read function.