
# ECS713U/ECS713P Functional Programming

## Individual Project

**Yash Raj Kumar Lahoti (210785180)**

## INTRODUCTION:

This report details the functionality of the Haskell programme that uses threads and performs concurrent operations.

## COMPILING AND RUNNING THE APP:

After downloading and extracting the project, execute the following command when navigated inside the folder: ***stack exec HaskellProject***.

## REQUIRED FUNCTIONALITIES:

This application generates 10 threads for 10 different users and counts the total number of messages that they have received. The application simulates 100 messages and random users send random messages across at random time intervals.

## EXTRA FUNCTIONALITIES:

An extra function to calculate the highest number of messages sent and to fetch the user details who has received the maximum number of messages has been created. This function can help us to identify which user has been using the social media platform for the maximum time. Details about the transaction has also been displayed in the Output. Each user has been allocated a unique message which is selected at random before they send it to another user.

## ISSUES FACED AND SOLUTION:

The output initially for counting the total number of messages received was of type (double). As we know, there can't be 1.5/2.5/... messages sent/received. Only an integer number of messages can be sent/received. To find the maximum and present it in the best possible way, it had to be converted to integer. That was done using: ***let tuser1count = round user1count :: Int*** , another way to do it is to change in **'Types.hs'** messages :: MVar Integer, it can save a lot of time and space, but it is not generalised as if there is a program which includes banking operations or computations, an integer won't help always.

## DESIGN CHOICE JUSTIFICATION:

The code has been kept as simple as possible and a very limited number of modules have been included, namely **Main.hs**, **SendMessage.hs**, **Types.hs**. I had initially kept MVar of type (Rational) – meaning, *double* thinking that it could be generalised for any kind of application, but if we see only about this system, it is better to use MVar of type (Int) to store the count of messages as it would then mean that only 1 transaction could take place across all the threads ensuring no interference with each other. MVars was used quite a lot of times in the program and it was the one which guided us to know what the actual count of the messages that are being received by each user. When randomising the receiver user, it was quite evident that we need to make the User datatype an instance of Eq class to enable direct comparison to check if random user = current user or no.

## HADDOCK DOCUMENTATION:

*stack exec -- haddock --html app/Main.hs src/SendMessage.hs src/Types.hs --hyperlinked-source --odir=dist/docs*

The haddock documentation is already generated and it can be found in "/dist/docs" directory.