COMP105: Programming Paradigms
Week 8 Homework Sheet

This is the homework sheet for **Week 8**. Complete your answers in a file named `week8.hs` and submit them to the "Week 8" assessment in SAM here

`https://sam.csc.liv.ac.uk/COMP/Submissions.pl`

Submission of the weekly homework sheets contributes 10% of the overall module mark, and each homework sheet counts equally towards this. Each homework sheet will be marked on a pass/fail basis. You will receive full marks for submitting a *reasonable attempt* at the homework. If no submission is made, or if a non-reasonable attempt is submitted, then no marks will be awarded.

The deadline for submission is

Friday Week 8 (04/12/2020) at 16:00.

Late submission is **not** possible. Individual feedback will not be given, but full solutions will be posted promptly after the deadline has passed.

If you feel that you are struggling with the homework, or if you have any other questions, then you can contact the lecturer at any point during the week via email, or you can drop in to the weekly Q&A session on MS Teams on Friday between 1PM and 4PM.

Lecture 23 – getLine and putStrLn.

1. Open ghci and run the `getLine` IO action. Type any string and press enter. Note that `getLine` returns the string that you entered.

2. Run `putStrLn "hello"`, and observe that it prints out `hello` with no quotes.

3. Copy the following code into your file

   ```
   echo :: IO ()
   echo = do
       str <- getLine
       putStrLn str
   ```

   Notice that we used a `do` block here, because we needed to unbox the result of `getLine` (which returns type `IO String`). **Make sure that the do block is indented** or you will get confusing errors later on. Run the `echo` action to check that it works.

4. Write an IO action `double_echo` that reads a string from the user, and then prints it out twice.

5. Write an IO action `put_two_strs` that takes two strings, and prints them both on different lines.

Lecture 23 – Let in do blocks Recall that we can use `let` in a `do` block like so

```
plus_one :: IO ()
plus_one = do
    str <- getLine
    let n   = read str :: Int
        out = n + 1
    putStrLn (show out)
```

The code above asks the user for a number, and then adds one to that number. Make sure that you understand this code before continuing.

1. Write an IO action `times_two` that asks the user for a number, and the prints out two-times that number.

2. Write an IO action `add` that asks the user for two numbers (on two different lines), and then prints out the sum of those two numbers.

3. Write an IO action `guess_42` that asks the user for a number. If the number is `42` then `correct` should be printed to the screen. Otherwise `wrong` should be printed.

Lecture 23 – Return. Recall that `return` lets us "box" a value in the IO type. Look at the following code

```
get_int :: IO Int
get_int = do
    str <- getLine
    let n = read str :: Int
    return n
```

The code asks the user for a number, converts it to an integer, and then returns that integer. Note that we needed to use `return`, in order to return `IO Int`, rather than `Int`. Make sure that you understand this code before continuing.

1. Write a function `get_bool :: IO Bool` that asks the user to input either `True` or `False` and returns the boolean value that they input. Remember that `read` can be used to parse `Bool`s.

2. Write a function `get_two_and_add :: IO Int` that asks the user for two integers, and returns the sum of those integers.

3. Write a function `get_two_strings :: IO (String, String)` that asks the user for two strings (on two different lines), and returns both strings that the user entered.

Lecture 24 – Looping in IO code. Recall that we can use recursion in IO code.

```
echo_forever :: IO ()
echo_forever = do
    str <- getLine
    putStrLn str
    echo_forever
```

The code above will continually ask the user for input, and then repeat that input, until the user presses control+c.

1. Write a function `add_one_forever` that continually asks the user for a number, and then prints out that number plus 1.
2. Write a function `echo_until_quit :: IO ()` that continually asks the user for input, and repeats that input, until the user enters `quit`.
3. (*) Write a function `print_numbers_between :: Int -> Int -> IO ()` that takes two numbers $a < b$ as arguments (so it doesn't ask the user for them), and prints out all the numbers between `a` and `b` (inclusive), each on a different line.

Lecture 24 – Compiling and running programs. Finally, let's practice compiling a Haskell program. Save the following code into a file named `prog.hs` (this is just for an example – there is no need to submit prog.hs)

```
main = putStrLn "Hello world!"
```

Windows users should open the Command Prompt (you can search for `cmd` to find it). By default, the current working directory will be something like `C:\Users\John`. First navigate to the folder in which you saved your program using the `cd` command.

```
cd C:\Users\John\COMP105\
```

If your folder contains a space then you will need to put the path in quotes.

```
cd "C:\Users\John\COMP 105\"
```

Then, type:

```
ghc prog
```

This will compile `prog.hs` creating the executable file `prog.exe`. You can then type:

```
prog
```

This will run the program, and print out `Hello world!`

OSX or Linux users should first open the Terminal program. Then use `cd` to change to the directory in which you saved your file.

```
cd /home/john/COMP105/
```

If your file name contains a space then you will need to put the path in quotes.

```
cd "/home/john/COMP 105/"
```

Then, type:

```
ghc prog
```

This will compile `prog.hs` creating the executable file `prog`. You can then type:

```
./prog
```

This will run the program, and print out `Hello world!`