Contents

1 Task 1

1.1 Extending token

First step is to extend the `Token` data type found in *Token.hs* file.

```
| Repeat    -- ^ \"repeat\"
| Until     -- ^ \"until\"
```

1.2 Adding keywords to the scanner

Second step is to have new keywords be picked up by our scanner, to do this we have to update `mkIdOrKwd` function in *Scanner.hs* file.

```
mkIdOrKwd "repeat" = Repeat
mkIdOrKwd "until" = Until
```

1.3 Adding new command to AST

We now have defined the token as well as we are able to recognize it using the scanner. The next step is to be able to represent it in out abstract syntax tree. To do so, we update `Command` data type inside `AST.hs`.

```
-- repeat until
| CmdRepeat {
      crCmd     :: Command,         -- ^ Action
      crCond    :: Expression,      -- ^ Condition
      cmdSrcPos :: SrcPos
  }
```

1.4 Keyword parser recognision

The next step is to update our parser to support this new syntax (all modification will be done to `Parser.y` file). First we modify the `token` to contain our new defined syntax

```
REPEAT      { (Repeat, $$) }
UNTIL       { (Until, $$) }
```

Next step is to add our command definition to the `command` function:

```
| REPEAT command UNTIL expression
    { CmdRepeat {crCmd = $2, crCond = $4, cmdSrcPos = $1} }
```

At this point our command will be picked up by the parser and the relative AST will be generated.

1.5 Pretty print

To allow debugging of the new syntax by printing, we have to add our command defined in *AST.hs* to *PPAST.hs* file. We add one more pattern match to `ppCommand`

```
ppCommand n (CmdRepeat {crCmd = cmd, crCond = cond, cmdSrcPos = sp}) =
    indent n . showString "CmdRepeat" . spc . ppSrcPos sp . nl
    . ppCommand (n+1) cmd
    . ppExpression (n+1) cond
```

Reference section

placeholder