## Contents

## 1 Task 1

## 2 Task 2

### 2.1 Repeat until

#### 2.1.1 MTIR

Update the MiniTriangle Internal Representation inside, so we can stored typed version

```
    -- | Repeat until
    | CmdRepeat {
          crCond    :: Expression,      -- ^ Loop-condition
          crBody    :: Command,         -- ^ Loop-body
          cmdSrcPos :: SrcPos
      }
```

### 2.2 TypeChecker

Add a pattern match for type checking AST `CmdRepeat` data type

```
-- T-REPEAT
chkCmd env (A.CmdRepeat {A.crCond = e, A.crBody = c, A.cmdSrcPos = sp}) = do
    e' <- chkTpExp env e Boolean                       -- env |- e : Boolean
    c' <- chkCmd env c                                 -- env |- c
    return (CmdRepeat {crCond = e', crBody = c', cmdSrcPos = sp})
```

### 2.3 PPMTIR

Now need a way to print the typed repeat command. We do this by adding a `CmdRepeat` pattern match to `ppCommand`

```
ppCommand n (CmdRepeat {crCond = e, crBody = c, cmdSrcPos = sp}) =
    indent n . showString "CmdRepeat" . spc . ppSrcPos sp . nl
    . ppCommand (n+1) c
    . ppExpression (n+1) e
```

### 2.4 Character literal

#### 2.4.1 Type

Firstly we add `Character` to Type data type

```
| Character           -- ^ The Character type
```

Next inside `instance Eq Type where` we add an equality operator pattern for it.

Character == Character = True

Finally, we add Character pattern match to `instance Show Type where`

```
showsPrec _ Character  = showString "Character"
```

#### 2.4.2 TypeChecker

We add a `ExpLitChar` pattern match to `infTpExp`. The only thing we do here is convert the character value to `MTChar` and transform $AST \rightarrow MTIR$

```
-- T-CHAR
infTpExp env e@(A.ExpLitChr {A.elcVal = c, A.expSrcPos = sp}) = do
    c' <- toMTChr c sp
    return (Character,                          -- env |- n : Character
            ExpLitChr {elcVal = c', expType = Character, expSrcPos = sp})
```