# Declarative characteristics

## Question 1

**Describe the parts of the eFLINT language used for defining relations (1 above), manipulating relations (2) and performing inference (3). Do not only list the constructs; describe how they are used to program with relations.**

Relations in eFlint are defined using the following key words:

- Fact: specifies an identity that has relations with other identities.
- Placeholder: does not directly define a relation, but is used later to define complex relations.
- Duty: Identifies a relation between two identities in which there is one party responsible for the duty and another party a claimant for that duty.
- Act: Defines a set of actions between 2 parties that create responsibilities for both parties.

```
Fact person
Placeholder parent    For person
Placeholder child     For person

Fact natural-parent   Identified by parent * child

Domain:
Fact person Identified by Alice, Bob, Chloe, David, Ellis
```

## Question 2

**In eFLINT you can write ?., i.e. a question mark followed by a Boolean expression followed by a full stop. Using the terminology of mathematical relations, logic programming, or database programming, what is this construct used for?**

In eFLINT this a boolean expression is used to validate if a Duty or Act has been performed according to the specifications.

A similar statement in SQL:

```
SELECT * FROM <TABLES> WHERE <BOOL-EXPR>

SELECT grades FROM programmeer-talen WHERE group_number=10
```

## Question 3

**What (non-syntactical) differences do you see between the language constructs discussed in the answers to Q1 and Q2 and similar constructs in Prolog or MySQL? Consider, for example, expressive**

**power,or behavior when executed. You can also compare with other logic programming or database languages if you prefer. Be specific.**

# Imperative characteristics

## Question 4

**Describe the parts of the eFLINT language that can be classified as imperative instructions. Explain how you decided that these are indeed the instructions of the language. For each kind of instruction, explain the effects these instructions can have.**

The 'scenario' part can be described as having an imperative style. Because it processes 'real-life' 'Duties', 'Acts', 'Events' and 'Observations' the order of steps is evaluated step by step, in an imperative manner.

## Question 5

**What would you say is the state manipulated by an eFLINT program? How does it change overtime? How does the manipulation of state help with assessing a normative (legal) case for compliance?**

# Semantic analysis

## Question 6

**Explain in detail when a duty is considered violated and in which ways a duty can become violated,for example the help-with-homework duty of the ‘Help with Homework’ example. In your explanation you can give and refer to examples. However, your explanation needs to be valid for all duties defined in the language.**

## Question 7

**Argue whether the language is statically or dynamically typed. How did you determine this?**

The language is dynamically typed. In the homework example the program runs the same in the following case:

```
-- Fact person Identified by String
Fact peron -- String identification removed

Domain:
-- Fact person Identified by Alice, Bob, Chloe, David
-- persons removed
```

This without type information it still functions.

## Question 8

**Write a C function that implements the ask-for-help act from the 'Help with Homework'example. The goal is that when this function is called with some arguments, the function behaves like when an instance of the ask-for-help type is triggered. Alternatively: use Haskell, Prolog, MySQL or pseudo-code instead of C.**

```
type Name = String
type Relation = Name
type ParentName = Name
type Child = (Name, ParentName)
type Holder = Name
type Claimant =  Name
type ChildName = Name
type Parent = (Name, ChildName)

isViolated :: (a -> Bool) -> a -> Bool
isViolated p x | p x  = True
               | otherwise = False

isLegalParent ::  Child -> Parent -> Bool
isLegalParent c p | fst c == snd p && fst p == snd c = True
                  | otherwise = False

createHomeworkDuty ::  Child -> Parent -> (Holder, Claimant, (a -> Bool) -
> a -> Bool)
createHomeworkDuty c p = (fst p, fst c, isViolated)

askForHelp :: Child -> Parent -> Maybe (Holder, Claimant, (a -> Bool) -> a
-> Bool)
askForHelp c p | isLegalParent c p = Just $ createHomeworkDuty c p
               | otherwise = Nothing
```