Introduction

Our program is split into 3 Haskell files. Main.hs provides the principal functionality of transforming JSON file into xml. DataTypes.hs provide the Haskell types used to convert JSON into haskell data. Finally, QueryDataFunctions.hs contains functions for querying parsed data.

JSON input

The source of the file used for this assignment is http://www.sitepoint.com/customer-form-json-file-example/. However, we slightly modified the file to make it more advanced and to allow more sophisticated data query functions. Our JSON file represents a customers array. Each customer has first name, last name, age, address and phoneNumbers as its attributes. Each address is a nested data structure including street address, city, state and postal code.The phone number's attribute is represented as an array of objects, each having number type and number itself as its attributes.

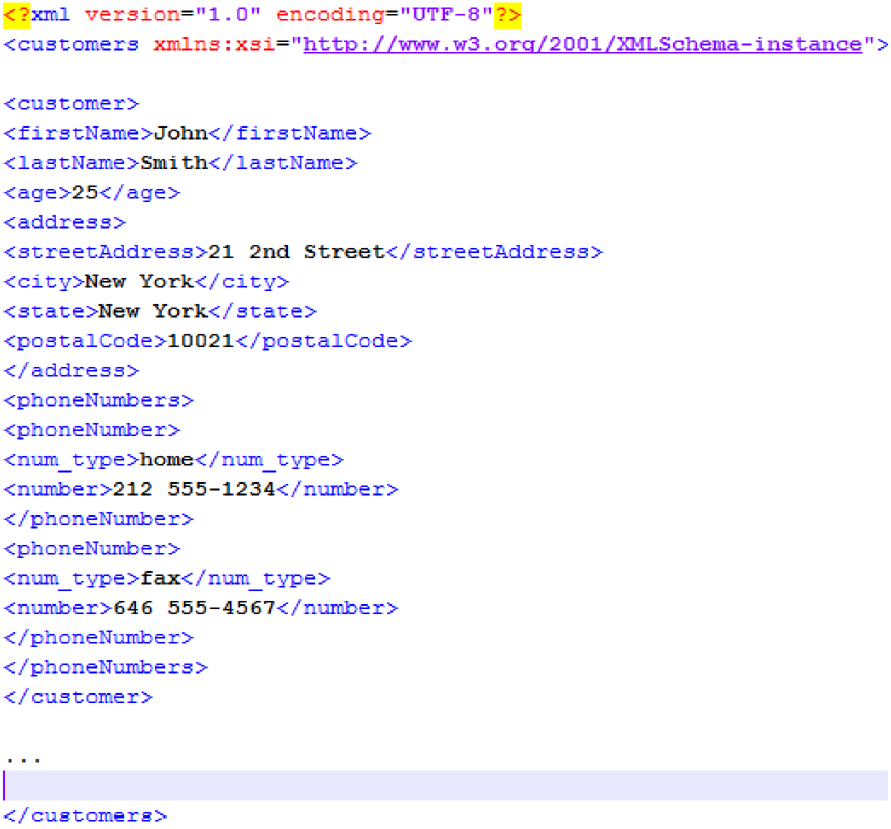
```
{
        "customers":[
                {
                        "firstName": "John",
                        "lastName": "Smith",
                        "age": 25,
                        "address":
                        {
                                "streetAddress": "21 2nd Street",
                                "city": "New York",
                                "state": "NY",
                                "postalCode": "10021"
                        },
                        "phoneNumbers":
                        [
                                {
                                 "num_type": "home",
                                 "number": "212 555-1234"
                                },
                                {
                                 "num_type": "fax",
                                 "number": "646 555-4567"
                                }
                        ]
                },


                ...
        ]
}
```

Haskell types to represent JSON data

```
data Customers = Customers
        { customers         :: [Customer]
        } deriving Show

data Customer = Customer
        { firstName    :: Text_M.Text
                , lastName :: Text_M.Text
                , age      :: Int
                , address  :: Address
                , phoneNumbers :: [PhoneNumber]
  } deriving Show

data Address = Address
        { streetAddress :: Text_M.Text
                , city                   :: Text_M.Text
```

```
                    , state       :: Text_M.Text
                    , postalCode:: Text_M.Text
             } deriving Show

data PhoneNumber = PhoneNumber
             { num_type          :: Text_M.Text
               ,number            :: Text_M.Text
             } deriving Show
```

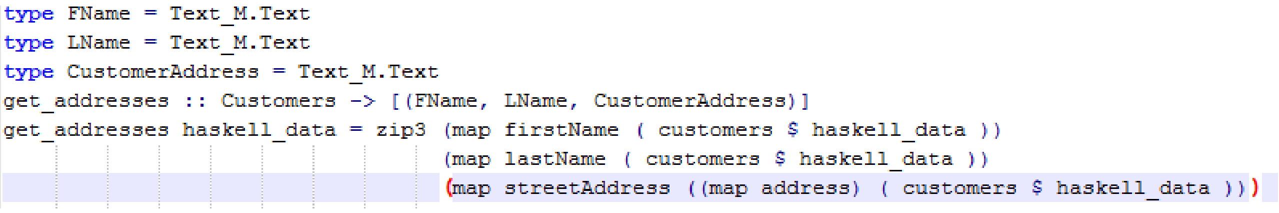
XML output



```
<?xml version="1.0" encoding="UTF-8"?>
<customers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<customer>
<firstName>John</firstName>
<lastName>Smith</lastName>
<age>25</age>
<address>
<streetAddress>21 2nd Street</streetAddress>
<city>New York</city>
<state>New York</state>
<postalCode>10021</postalCode>
</address>
<phoneNumbers>
<phoneNumber>
<num_type>home</num_type>
<number>212 555-1234</number>
</phoneNumber>
<phoneNumber>
<num_type>fax</num_type>
<number>646 555-4567</number>
</phoneNumber>
</phoneNumbers>
</customer>

...

</customers>
```

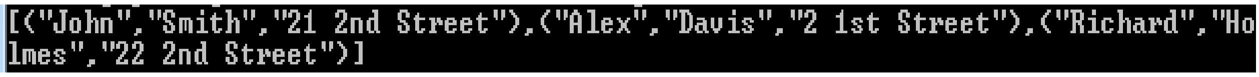
Query functions

Our query functions are: get_age, avg_customer_age, min_customer_age, get_addresses and cnt_cust_in_postcode. For instance, get_addresses functions zips together customer names and their addresses:



```
type FName = Text_M.Text
type LName = Text_M.Text
type CustomerAddress = Text_M.Text
get_addresses :: Customers -> [(FName, LName, CustomerAddress)]
get_addresses haskell_data = zip3 (map firstName ( customers $ haskell_data ))
                                  (map lastName ( customers $ haskell_data ))
                                  (map streetAddress ((map address) ( customers $ haskell_data )))
```

Terminal output (when run on $ get_haskell json_in):



```
[("John","Smith","21 2nd Street"),("Alex","Davis","2 1st Street"),("Richard","Ho
lmes","22 2nd Street")]
```