```haskell
-- vismol2-1.hs

import Data.IORef
import Graphics.UI.GLUT
import Graphics.Rendering.OpenGL.GLU.Quadrics
import Data.List (isPrefixOf)

main :: IO ()
main = do
    m <- getMolecule "yperite.mol2"
    visualize $ moveMolecule m 5.0 5.0 5.0 -- to test out the centering

getMolecule :: FilePath -> IO Molecule
getMolecule filePath = do
    theText <- readFile filePath
    return (getRecords (lines theText) OtherRecord [] [])

data Molecule = Molecule [Atom] [Bond] deriving (Show)
data Atom = Atom Int Float Float Float String deriving (Show) -- i x y z type
data Bond = Bond Int Int String deriving (Show) -- i j type

data RecordType = AtomRecord | BondRecord | OtherRecord

getRecords :: [String] -> RecordType -> [Atom] -> [Bond] -> Molecule
getRecords ("@<TRIPOS>ATOM":ls) _ atoms bonds = getRecords ls AtomRecord atoms bonds
getRecords ("@<TRIPOS>BOND":ls) _ atoms bonds = getRecords ls BondRecord atoms bonds
getRecords (('@':_):ls) _ atoms bonds = getRecords ls OtherRecord atoms bonds
getRecords (_:ls) OtherRecord atoms bonds = getRecords ls OtherRecord atoms bonds
getRecords (l:ls) AtomRecord atoms bonds =
    getRecords ls AtomRecord (atom:atoms) bonds where
        atom = Atom i x y z t where
            fields = words l
            i = read $ fields !! 0
            x = read $ fields !! 2
            y = read $ fields !! 3
            z = read $ fields !! 4
            t = fields !! 5
getRecords (l:ls) BondRecord atoms bonds =
    getRecords ls BondRecord atoms (bond:bonds) where
        bond = Bond i j t where
            fields = words l
            i = read $ fields !! 1
            j = read $ fields !! 2
            t = fields !! 3
getRecords [] _ atoms bonds = Molecule atoms bonds

moveMolecule :: Molecule -> Float -> Float -> Float -> Molecule
moveMolecule m@(Molecule atoms bonds) dx dy dz =
    Molecule ( map (\a -> moveAtom a dx dy dz) atoms ) bonds

moveAtom :: Atom -> Float -> Float -> Float -> Atom
moveAtom a@(Atom i x y z t) dx dy dz =
    Atom i (x+dx) (y+dy) (z+dz) t

data Axis = Ox | Oy | Oz

rotateMolecule :: Molecule -> Axis -> Float -> Molecule
rotateMolecule m@(Molecule atoms bonds) aroundAxis angle =
    Molecule ( map (\a -> rotateAtom a aroundAxis angle) atoms ) bonds

rotateAtom :: Atom -> Axis -> Float -> Atom
rotateAtom a@(Atom i x y z t) Ox angle =
    Atom i x y' z' t where
        y' =  y * (cos angle) + z * (sin angle)
        z' = -y * (sin angle) + z * (cos angle)
rotateAtom a@(Atom i x y z t) Oy angle =
    Atom i x' y z' t where
        x' = x * (cos angle) - z * (sin angle)
        z' = z * (sin angle) + z * (cos angle)
rotateAtom a@(Atom i x y z t) Oz angle =
    Atom i x' y' z t where
        x' =  x * (cos angle) - y * (sin angle)
        y' = -x * (sin angle) + y * (cos angle)

```

```haskell
visualize m@(Molecule atoms bonds) = do
    -- initialization and window
    getArgsAndInitialize
    initialDisplayMode $= [ DoubleBuffered, RGBMode, WithDepthBuffer ]
    initialWindowSize $= Size 300 300
    initialWindowPosition $= Position (-1) (-1)
    createWindow "Zzzz!.."
    -- unchanging attributes of the scene
    clearColor $= Color4 0 0 0 1
    shadeModel $= Smooth
    materialAmbient Front $= Color4 1 1 1 1
    lighting $= Enabled
    position (Light 0) $= Vertex4 (-1) 1 1 0
    light (Light 0) $= Enabled
    depthFunc $= Just Less
    materialDiffuse Front $= Color4 1 1 1 1
    -- callbacks
    displayCallback $= (display $ moveMolecule m (-cx) (-cy) (-cz))
    reshapeCallback $= Just (reshape sz)
    -- keyboardMouseCallback $= (Just keyboardMouse)
    mainLoop
    where
        maxx = maximum $ map (\a@(Atom _ x _ _ _) -> x) atoms
        maxy = maximum $ map (\a@(Atom _ _ y _ _) -> y) atoms
        maxz = maximum $ map (\a@(Atom _ _ _ z _) -> z) atoms
        minx = minimum $ map (\a@(Atom _ x _ _ _) -> x) atoms
        miny = minimum $ map (\a@(Atom _ _ y _ _) -> y) atoms
        minz = minimum $ map (\a@(Atom _ _ _ z _) -> z) atoms
        sz = realToFrac $ 1.05 * maximum [maxx-minx, maxy-miny, maxz-minz]
        cx = (sum $ map (\a@(Atom _ x _ _ _) -> x) atoms) / (fromIntegral $ length atoms)
        cy = (sum $ map (\a@(Atom _ _ y _ _) -> y) atoms) / (fromIntegral $ length atoms)
        cz = (sum $ map (\a@(Atom _ _ _ z _) -> z) atoms) / (fromIntegral $ length atoms)

display m@(Molecule atoms bonds) = do
    print "-- display"
    clear [ ColorBuffer, DepthBuffer ]
    renderAtoms atoms
    renderBonds atoms bonds
    swapBuffers

renderAtoms (a@(Atom _ x y z t):atoms) = do
    materialDiffuse Front $= atomColor4 t
    translate $ Vector3 x y z
    renderObject Solid $ Sphere' (radiusOfAtom t) 32 32
    translate $ Vector3 (-x) (-y) (-z)
    renderAtoms atoms
renderAtoms [] = do return () -- do nothing

atomColor4 atomType
    | atomType == "H"  = Color4 1 1 1 1 -- white
    | atomType == "F"  = Color4 0 1 0 1 -- green
    | atomType == "Cl" = Color4 0 1 0 1 -- green
    | atomType == "Br" = Color4 0 1 0 1 -- green
    | atomType == "I"  = Color4 0 1 0 1 -- green
    | isPrefixOf "C." atomType = Color4 0 1 1 1 -- cyan instead of gray
    | isPrefixOf "N." atomType = Color4 0 0 1 1 -- blue
    | isPrefixOf "O." atomType = Color4 1 0 0 1 -- red
    | isPrefixOf "S." atomType = Color4 1 1 0 1 -- yellow
    | isPrefixOf "P." atomType = Color4 1 0 1 1 -- magenta
    | otherwise = Color4 1 0 1 1 -- magenta

radiusOfAtom atomType
    | atomType == "H"  = 0.38
    | otherwise = 0.62

renderBonds atoms (b@(Bond i j _):bonds) = do
    materialDiffuse Front $= Color4 1 1 1 1
    --translate $ Vector3 x y z
    renderQuadric style $ Cylinder 1.0 0.24 1.0 32 1 -- not from GLUT
    --translate $ Vector3 (-x) (-y) (-z)
    renderBonds atoms bonds
    where

        style = QuadricStyle (Just Smooth) NoTextureCoordinates Outside FillStyle
```

```haskell
renderBonds _ [] = do return () -- do nothing

reshape sz size@(Size width height) = do
    print "--reshape"
    print width
    print height
    --
    viewport $= (Position 0 0, size)
    matrixMode $= Projection
    loadIdentity
    if width <= height
        then ortho (-sz) sz (-sz * h/w) (sz * h/w) (-sz) sz
        else ortho (-sz * w/h) (sz * w/h) (-sz) sz (-sz) sz
    matrixMode $= Modelview 0
    loadIdentity
    where
        w = fromIntegral width
        h = fromIntegral height

-- keyboardMouse _ _ (Char '\27') Down _ _ = exitWith ExitSuccess -- exit on ESC
```