```
type Scanner a = GenParser Char () a
data Token
    =   INum Integer | FNum Double
      | Varid String | Reserved String
    deriving (Show, Eq)
scan :: Scanner a -> Scanner (a, SourcePos)
scan p = do  pos <- getPosition
             x <- p
             return (x, pos)
scan_integer, scan_varid, hreserved, htoken :: Scanner (Token, SourcePos)
scan_integer = do  (i, pos) <- scan (integer (makeTokenParser haskellDef))
                   return (INum i, pos)
hfloat = do  (i, pos) <- scan (float (makeTokenParser haskellDef))
             return (FNum i, pos)
scan_varid = do  (c, pos) <- scan (hlower)
                 cs <- identifier (makeTokenParser haskellDef)
                 return (Varid (c : cs), pos)
hlower :: Scanner Char
hlower = lower <|> char '_'
hreserved = do  (cs, pos) <- scan (string "(" <|> string ")")
                return (Reserved cs, pos)
htoken = try hfloat <|> scan_integer <|> scan_varid <|> hreserved
scanall :: String -> [(Token, SourcePos)]
scanall cs = let result = parse (many htoken) "" cs
             in case result of
                  Right tkns -> tkns
                  Left err -> error (show err)
```