Дәріс № 7

Монадалар

Функционалды бағдарламалауда жаңадан бастаушылар Haskell' дегі монада түсінігін түсіне бермейді. Алайда монадалар тілде жиі кездеседі, мысалы енгізу\шығару жүйесі монада түсінігіне негізделген. Ал стандартты кітапханаларда монадаларға арналған бүтін бір модульдар бар. Монада түсінігі Haskell де категориялар теориясына негізделгенін айтып өткен жөн , алайда абстракциялы математикаға кіріспеу үшін монадалардың интуитивті түсінігі келтіріледі.

Монадалар келесі монадалық кластардың түрлеріне жататын типтер болып табылады: Functor, Monad и MonadPlus. Бұл кластардың бірі де басқа класқа тегі бола ламайды себебі монадалық кластар мұраға қалдырылмайды.

Prelude модулінде үш монада анықталған: IO, [] және Maybe, яғни тізім де монада болып табылады.

Монада математикалық түрде монадаға орындалатын операцияларды байланыстырып тұрған ережелер жиынтығымен анықталады. Бұл ережелер монаданы қалай қолдану керектінің және оның ішкі құрылымы туралы түсінік береді. Нақтырақ түсіну үшін ары қарай екі базалық операциясы және бір функциясы анықталған Monada класы қарастырылады:

- class Monad m where
- (>>=) :: m a -> (a -> m b) -> m b
- (>>) :: m a -> m b -> m b
- return :: a -> m a
- fail :: String -> m a
- m >> k = m >>= _ -> k

Екі операциялар (>>=) және (>>) — бұл байланыстыру операциялары. Олар екі монадалық мәнді біріктіреді ал return функциясы a типті берілген мәнді m a типті монадалық мәнге ауыстырады. Сигнатуралық операциялар (>>) байланыстыру операцияларын түсінуге көмектемеді: (m a >>= \v → m b) түсінігі a типті нысанды v типті функциямен біріктіріп m b типтерінің нәтижесін қайтаратын m a монадалық мәнін біріктіреді. Нәтижесі m b типті монадалық мән болады. (>>) операциясы бірінші монадалық операндтан алынған мәнді функция қолданбаған жағдайда қолданылады. Байланыстыру операциясының нақты мәні әрине монаданы нақты іске асыруға тәуелді

Мысалы, IO типі (>>=) операциясын оның екі операндының тізбектеле орындалғаны, яғни орындалған бірінші операнданың мәні екіншісіне берілуі ретінде қарастырылады. Басқа қосымша екі монадалық типтерге (тізімдер және Maybe) бұл операция нөлдің берілгені ретінде немесе одан көп параметрлерді есептеу процесінен келесіге беру ретінде анықталады.

Haskell'де тіл деңгейінде монаданы қолдануды қолдайтын арнайы қызметші сөз болады. Бұл do сөзі оның түсінігін мына қодану ережелерінен көруге болады:

```
do e1 ; e2 = e1 >> e2
do p <- e1 ; e2 = e1 >>= \p -> e2
```

Біріншісі әрқашан орнылады (бірінші операндадан екіншісіне ауыстыру жасалмайды). Екіншісінде қате кетуі мүмкін, ол жағдайда Monada класында анықталған fail функциясы шақырылады.Сондықтан do сөзінің нақтырақ анықтамасы былай жазылады:

```
do p <- e1 ; e2 = e1 >>= (\v -> case v of
p -> e2
_ -> fail ”s”)
```

s — бұл жол, ол do операторының бағдарламада орналасуын анықтай алады, немесе біршама семантикалық жүктемені алуы мүмкін. Мысалы, IO монадасында (‘a’ ← getChar) әрекеті fail функциясын тек оқылған символ ‘a’ символы болмаған жағдайда ғана шақырады.

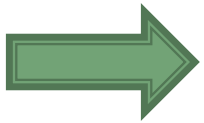
Бұл әрекет бағдарламаның орындалуын үзеді, себебі IO монадасында fail функциясы өз кезегінде error жүйелік функциясын шақырады.

MonadPlus класы нөлдік элементі бар және «+» операциясы бар монадалар үшін қолданылады. Бұл кластың анықтамасы келесідей:

```
class (Monad m) => MonadPlus m where
mzero :: m a
mplus :: m a -> m a -> m a
```

Бұл монадалар класындағы нөлдік элемен келесі ережелерге бағынады:

```
m >>= \x -> mzero = mzero
mzero >>= m = mzero
```



Мысалы, тізімдер үшін нөлдік элемент болып [] бос тізім, ал «+» операциясына — тізімдер конкатенациясы. Сондықтан тізімдер монадасы MonadPlus класының түрі болып табылады. Басқа жағынан қарағанда IO монадасында нөлдік элемент жоқ, сондықтан IO монадасы тек Monad класына жатады.

Қосымша монадалар

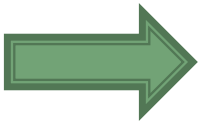
Алынған ақпараттарды нақтылау үшін қарапайымдау етіп қарастыру керек. Тізімдер монада болып табылатындықтан және сол тізімдер жан жақты қарастырылғандықтан олардың мысалында монадалардың іс жүзінде қолданылуын көруге болады.

------Тізімдер үшін байланыстыру операциясы тізімнің әр элементі үшін орындалатын операциялар жинағы ретінде қарастырылады.Тізімдермен бірге қолданылатын (>>=) операциясының сигнатурасы келесі түрге келеді:

(>>=) :: [a] -> (a -> [b]) -> [b]

Бұл a типті мәні бар тізім және a типті мәнді b типті мәнге келтіретін функция берілгенін білдіреді. Байланыстыру a типті мәні бар тізімнің әр элементке функцияны қолданып, b типті мән тізіміне қайтарады. Бұл операция бізге белгілі — тізімдерді анықтаушылар дәл осы жолмен анықтайды. Яғни келесі үш өрнек бірдей болады:

```
Prelude>:set +t
Prelude>1
1 :: Integer
Prelude>1.2
1.2 :: Double
Prelude>'a'
'a' :: Char
Prelude>True
True :: Bool
```



```
-- Өрнек 1 ----------------------------
[(x, y) | x <- [1, 2, 3], y <- [1, 2, 3], x /= y]
-- Өрнек 2----------------------------------
do x <- [1, 2, 3]
y <- [1, 2, 3]
True <- return (x /= y)
return (x, y)
-- Өрнек 3----------------------------------
[1, 2, 3] >>= (\x -> [1, 2, 3] >>= (\y -> return (x /= y)
>>=
(\r -> case r of
True -> return (x, y)
_ -> fail ””)))
```

Қай өрнекті қолдануды бағдарламалаушы шешеді.