Функційне програмування мовою Haskell

Монади

Функційне програмування мовою Haskell

Монади

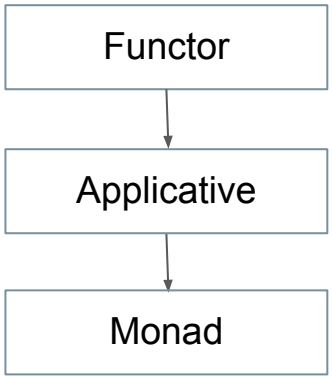
Обчислення у контекстах

Класи:

- Functor
- Applicative (аплікативний функтор)
- Monad

Кажуть про:
контейнерні типи;
обчислювальний контекст

- [] недетермінований результат
- Maybe з можливим неуспіхом
- Either a з можливим неуспіхом та повідомленням
- IO з можливим побічним ефектом
- Tree ієрархічна структура



Functor f

fmap :: (a -> b) -> f a -> f b

Functor f

```
fmap ::  (a -> b) ->  f a ->  f b
(<$>) :: (a -> b) -> f a -> f b
```
(інфіксний оператор - синонім fmap)

Functor f

fmap :: (a -> b) -> f a -> f b
(<$>) :: (a -> b) -> f a -> f b (інфіксний оператор - синонім fmap)

> (*10) <$> [1..3]

> (*10) <$> Just 2

> (*10) <$> Nothing

Functor **f**

```
fmap ::  (a -> b) ->  f a ->  f b
(<$>) :: (a -> b) -> f a -> f b     (infixl 4 - синонім fmap)

(<$)   ::  a -> f b -> f a          (infixl 4)
```

Functor **f**

```
fmap ::  (a -> b) ->  f a ->  f b
(<$>) :: (a -> b) -> f a -> f b    (infixl 4 - синонім fmap)
(<$)   ::  a -> f b -> f a          (infixl 4)

> (<$) 2  [1..3]

> (<$) 2  []

> (<$) 2  Nothing

> (<$) 2  (Just 10)
```

```
class Functor f where

fmap ::  (a -> b) ->  f a ->  f b
(<$>) :: (a -> b) -> f a -> f b     (infixl 4 - синонім fmap)
(<$)   ::  a -> f b -> f a            (infixl 4)


class Functor f => Applicative f where
   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
```

Functor **f**

```
fmap ::  (a -> b) ->  f a ->  f b
(<$>) :: (a -> b) -> f a -> f b    (infixl 4 - синонім fmap)
(<$)  ::  a -> f b -> f a          (infixl 4)
```

```
class Functor f => Applicative f where
   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
```

```
class Functor f => Applicative f where
   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b

> pure 3 :: Maybe Int

> pure 3 :: [Int]

> pure 3 :: Either String Int
```

```
class Functor f => Applicative f where
   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
```

```
class Functor f => Applicative f where
   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b

> pure (*10) <*> Just 2

> pure (*10) <*> Nothing

> pure (*10) <*> [1..3]

> [(*2),(+100)] <*> [10,20]

> pure (*10) <*> Right 2
```

```
class Functor f => Applicative f where
   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b

> pure (+) <*> Just 2 <*> Just 3

> pure (+) <*> Just 2 <*> Nothing

> pure (+) <*> [1..3] <*> [10,20]

> pure (+) <*> Right 2 <*> Right 3

> Right (+) <*> Right 2 <*> Right 3

>  (+) <$> Right 2 <*> Right 3
```

```
class Functor f => Applicative f where
   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
```

Послідовність дій. Відкидаються значення першого (другого) аргументу.

```
(*>) :: f a -> f b -> f b
(<*) :: f a -> f b -> f a
```

```
class Functor f => Applicative f where
   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
```

Послідовність дій. Відкидаються значення першого (другого) аргументу.

```
(*>) :: f a -> f b -> f b
(<*) :: f a -> f b -> f a
```

```
> (Just 2) *> (Just 3)
```

```
> Nothing *> (Just 3)
```

```
> undefined *> (Just 3)
```

```
> [1,2,3] *> [10,20]
```

```
> [] *> [10,20]
```

```
class Functor f => Applicative f where
   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
```

Послідовність дій. Відкидаються значення першого (другого) аргументу.

```
(*>) :: f a -> f b -> f b
(<*) :: f a -> f b -> f a
```

```
> (Just 2) *> (Just 3)

> Nothing *> (Just 3)

> undefined *> (Just 3)

> [1,2,3] *> [10,20]

> [] *> [10,20]
```

```
> (Just 2) <* (Just 3)

> Nothing <* (Just 3)

> undefined <* (Just 3)

> [1,2,3] <* [10,20]

> [] <* [10,20]
```

```
class Functor f where
  fmap ::  (a -> b) ->  f a ->  f b
  (<$>) :: (a -> b) -> f a -> f b
  (<$)   ::  a -> f b -> f a

class Functor f => Applicative f where
   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
  (*>) :: f a -> f b -> f b
  (<*) :: f a -> f b -> f a

class Applicative m => Monad m where
  return :: a -> m a
  (>>=)  :: m a -> (a -> m b) -> m b
  (>>)    :: m a -> m b -> m b
  fail :: String -> m a
```

(синонім fmap)

```
class Functor f where
  fmap ::  (a -> b) ->  f a ->  f b
  (<$>) :: (a -> b) -> f a -> f b     (синонім fmap)
  (<$)   ::  a -> f b -> f a

class Functor f => Applicative f where
   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
  (*>) :: f a -> f b -> f b
  (<*) :: f a -> f b -> f a

class Applicative m => Monad m where
  return :: a -> m a
  (>>=)  :: m a -> (a -> m b) -> m b
  (>>)    :: m a -> m b -> m b
  liftM    :: (a -> r) -> m a -> m r          – модуль Control.Monad
  ap :: m (a -> b) -> m a -> m b              – модуль Control.Monad
```

```
class Functor f where
  fmap ::  (a -> b) ->  f a ->  f b
  (<$>) :: (a -> b) -> f a -> f b     (синонім fmap)
  (<$)   ::  a -> f b -> f a

class Functor f => Applicative f where
   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
  (*>) :: f a -> f b -> f b
  (<*) :: f a -> f b -> f a

class Applicative m => Monad m where
  return :: a -> m a
  (>>=)  :: m a -> (a -> m b) -> m b
  (>>)    :: m a -> m b -> m b
  liftM    :: (a -> r) -> m a -> m r          – модуль Control.Monad
  ap :: m (a -> b) -> m a -> m b              – модуль Control.Monad
```

```
class Functor f where
  fmap ::  (a -> b) ->  f a ->  f b
  (<$>) :: (a -> b) -> f a -> f b    (синонім fmap)
  (<$)   ::  a -> f b -> f a

class Functor f => Applicative f where
   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
  (*>) :: f a -> f b -> f b
  (<*) :: f a -> f b -> f a

class Applicative m => Monad m where
  return :: a -> m a
  (>>=)  :: m a -> (a -> m b) -> m b
  (>>)    :: m a -> m b -> m b
  liftM    :: (a -> r) -> m a -> m r          – модуль Control.Monad
  ap :: m (a -> b) -> m a -> m b              – модуль Control.Monad
```

```
class Functor f where
  fmap ::  (a -> b) ->  f a ->  f b
  (<$>) :: (a -> b) -> f a -> f b    (синонім fmap)
  (<$)   ::  a -> f b -> f a

class Functor f => Applicative f where
   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
  (*>) :: f a -> f b -> f b
  (<*) :: f a -> f b -> f a

class Applicative m => Monad m where
  return :: a -> m a
  (>>=)  :: m a -> (a -> m b) -> m b
  (>>)    :: m a -> m b -> m b
  liftM    :: (a -> r) -> m a -> m r        – модуль Control.Monad
  ap :: m (a -> b) -> m a -> m b            – модуль Control.Monad
```

liftM дозволяє застосовувати звичайну функцію до монадичного значення без необхідності do-блоків

```
> import Control.Monad
>  liftM (*2) [1..3]

>  liftM (*2) (Just 3)

>  (<$>) (*2) [1..3]

>  (<$>) (*2) (Just 3)
```

```
class Functor f where
  fmap ::  (a -> b) ->  f a ->  f b
  (<$>) :: (a -> b) -> f a -> f b    (синонім fmap)
  (<$)   ::  a -> f b -> f a

class Functor f => Applicative f where
   pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b          --
  (*>) :: f a -> f b -> f b
  (<*) :: f a -> f b -> f a

class Applicative m => Monad m where
  return :: a -> m a
  (>>=)  :: m a -> (a -> m b) -> m b
  (>>)    :: m a -> m b -> m b
  liftM    :: (a -> r) -> m a -> m r        – модуль Control.Monad
  ap :: m (a -> b) -> m a -> m b            -- модуль Control.Monad
```

```
> Just (*10) `ap` Just 3

> Just (*10) `ap` Nothing

> return (*10) `ap` [1..3]

> [(*10)] `ap` [1..3]

> [(*10)] <*> [1..3]

> [] `ap` [1..3]

> return (*) `ap` [1..3] `ap` [10,20]

> liftM (*)  [1..3] `ap` [10,20]

> (*) <$>  [1..3] <*> [10,20]
```

```
> Just 10 >>= \x -> Just (x*2)

> Just 10 >>= \x -> Just (x*2) >>= \y -> Just (y+100)

> Nothing >>= \x -> Just (x*2) >>= \y -> Just (y+100)
```

```
> Just 10 >>= \x -> Just (x*2) >>= \y -> Just (y+100)

Prelude> :{
Prelude| f1 :: Maybe Int
Prelude| f1 = Just 10 >>= \x ->
Prelude|      Just (x*2) >>= \y ->
Prelude|      Just (y+100)
Prelude| :}
Prelude> f1
```

```
> Just 10 >>= \x -> Just (x*2) >>= \y -> Just (y+100)
```

```
Prelude> :{
Prelude| f1 :: Maybe Int
Prelude| f1 = Just 10 >>= \x ->
Prelude|      Just (x*2) >>= \y ->
Prelude|      Just (y+100)
Prelude| :}
Prelude> f1
```

```
Prelude> :{
Prelude| f2 :: Maybe Int
Prelude| f2 = do
Prelude|    x <- Just 10
Prelude|    y <- Just (x*2)
Prelude|    Just (y+100)
Prelude| :}
Prelude> f2
```

[do notation]