Декларативное программирование

Real World

Hello, World!

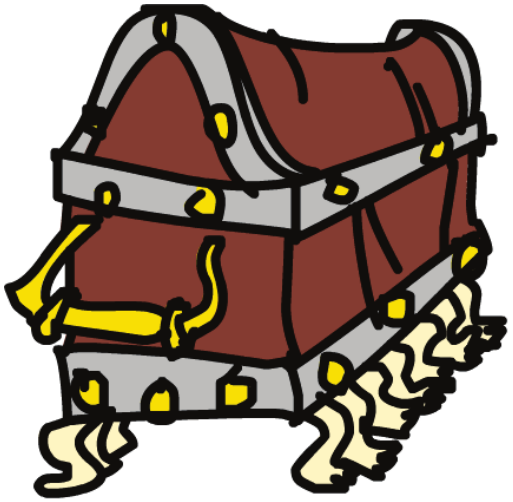
```
main = putStrLn "Hello, World"
```

helloworld.hs

```
>ghc helloworld.hs
>./helloworld
```

Объединение действий ввода-вывода

```
main = do
    putStrLn "Hello, what is your name?"
    name <- getLine
    putStrLn ("Hello, " ++ name ++ ", nice to meet you!")
```



let

```
import Data.Char
main = do
    putStrLn "Ваше имя?"
    firstName <- getLine
    putStrLn "Ваша фамилия?"
    lastName <- getLine
    let bigFirstName = map toUpper firstName
        bigLastName = map toUpper lastName
    putStrLn $ "Привет, " ++ bigFirstName ++ " " ++ bigLastName ++ ", как дела?"
```

return

создаёт действие ввода-вывода из чистого значения

```
main = do
    let a = "ад"
        b = "да!"
    putStrLn $ a ++ " " ++ b
```

```
main = do
    a <- return "ад"
    b <- return "да!"
    putStrLn $ a ++ " " ++ b
```

return

```
produceTwo :: IO Int
produceTwo = do return 1
                return 2
```

Повторяющиеся действия

```
printList :: [Int] -> IO ()
printList [] = return ()
printList (x:xs) = do
                   print x
                   printList xs
```

Control.Monad

```
-- выполнить операцию, если предикат истинен
when :: Bool -> IO () -> IO ()
-- выполнить операцию, если предикат ложен
unless :: Bool -> IO () -> IO ()
-- выполнить несколько раз, сохранить результаты
replicateM :: Int -> IO a -> IO [a]
-- выполнить несколько раз, игнорировать результаты
replicateM_ :: Int -> IO a -> IO ()
-- выполнить для каждого элемента списка, сохранить результаты
mapM :: (a -> IO b) -> [a] -> IO [b]
-- выполнить для каждого элемента списка, игнорировать результаты
mapM_ :: (a -> IO b) -> [a] -> IO ()
-- то же самое, другой порядок аргументов
forM  :: [a] -> (a -> IO b) -> IO [b]
forM_ :: [a] -> (a -> IO b) -> IO ()
```

Упражнение

Определите действие ввода-вывода readUntil f, которое читает строки от пользователя и возвращает их в виде списка. Чтение прекращается, когда f возвращает True для какой-либо строки. (Значение, для которого f возвращает True, не возвращается).

```
readUntil :: (String -> Bool) -> IO [String]
readUntil f = undefined
```

Упражнение

Реализовать цикл while, он должен выполнять операцию до тех пор, пока условие возвращает True.

```
while :: IO Bool -> IO () -> IO ()

while cond op = undefined
```

Упражнение

Напишите программу, которая будет считывать строки, переставлять в обратном порядке буквы в словах и распечатывать их. Выполнение программы должно прекратиться при вводе пустой строки.

```
reverseWords :: String –> String

reverseWords = unwords . map reverse . words
```

Упражнение

Какой тип у этой функции?

```
foo x = do putStrLn x
          y <- getLine
          return (length y)
```

IORef

```
-- создание
newIORef :: a -> IO (IORef a)
-- чтение
readIORef :: IORef a -> IO a
-- запись
writeIORef :: IORef a -> a -> IO ()
-- изменение
modifyIORef :: IORef a -> (a -> a) -> IO ()
-- строгая версия
modifyIORef' :: IORef a -> (a -> a) -> IO ()
```

IORef

```
testIORef = do
    ref <- newIORef 1
    val1 <- readIORef ref
    writeIORef ref 41
    val2 <- readIORef ref
    modifyIORef' ref succ
    val3 <- readIORef ref
    return [val1,val2,val3]
```

Упражнение

Напишите функцию в «императивном» стиле, вычисляющую факто-
риал с использованием IORef.

```
fac'' :: Integer -> IO Integer
fac'' n = do undefined
```

Случайные числа (System.Random)

Два способа получить генератор псевдо-случайных чисел:

1. Использовать глобальный, инициализированный системным временем

```
GHCi> :t getStdGen
getStdGen :: IO StdGen
GHCi> getStdGen
1033221633 1
```

2. Если есть требование воспроизводимости — создать свой:

```
GHCi> :t mkStdGen
mkStdGen :: Int -> StdGen
GHCi> myGen = mkStdGen 42
GHCi> myGen
43 1
```

Случайные числа (System.Random)

```
randomIO :: IO a
random :: RandomGen g => g -> (a, g)
randoms :: RandomGen g => g -> [a]
```

Упражнение

Напишите программу, которая эмулирует случайный эксперимент с использованием генератора случайных чисел. Эксперимент заключается в наблюдении за k сериями подбрасываний монетки. Серия состоит из n подбрасываний. Каждое подбрасывание возвращает либо "орёл" (H), либо "решка" (T) с равной вероятностью.