ソフトウェア科学特論第2回(その2)

目次

参考にしたサイト

- qiita:ハスカル超入門
- とほほのハスカル入門

関数

関数は引数を受け取り、処理を行い結果を返すものです。

- 関数定義の例

```
  add x y = x + y
```

- 関数呼び出し

```
main = print (add 3 5)
```

演算子定義

演算子を定義することができ、infixl(左結合)、infixr(右結合)、infix(結合無し)を用いて演算子の優先度を指定することができます。

- 演算子定義の例

```
x ^^^ y = x * 1000 + y
main = print $ 2 ^^^ 20
```

- 演算子の優先度設定

```
infixl 7 +++
infixl 6 ***
```

再帰関数

関数が自分自身を呼び出すことによって定義される関数のことで、再帰的な処理を実現します。

- 階乗を求める関数の例

```
fact 0 = 1
fact n = n * fact (n - 1)main = print $ fact 5
```

ラムダ式

関数名を持たない、式を定義することができる局所関数であり、\arg -> exprという形式で表されます。

- ラムダ式の例

```
main = do
  print c
    where
      c = a + b
      a = (\x -> x * x) 5
      b = (\(x, y) -> x * y) (2, 3)
```

パターンマッチ

パターンマッチを使用すると、関数の引数の値によって関数を別々に定義することができます。

- 関数の引数による定義

```
func 1 = "One"func 2 = "Two"func 3 = "Three"main = print $ func 1
```

ガード条件

関数をパターンマッチと似た構文で定義することができ、条件によって異なる式を返すことができます。

- ガード条件を用いた関数定義

```
foo x
  | x == 1 = "One"
  | x == 2 = "Two"
  | x == 3 = "Three"
  | otherwise = "More..."
main = putStrLn $ foo 2
```

関数合成(.)

演算子を用いて、複数の関数を合成することができます。

- 関数合成の例

```
fn n = (f . g . h) n
```

引数補足(@)

@を用いて、引数を複数の形式で受け取ることができます。

- 引数補足の例

```
func str@(x:xs) = do
  print str
  print x
  print xs
main = do
  func "ABCDE"
```