プログラミング基礎 実習資料 第 02 時限

1 Haskell プログラム

コンピュータに問題を解決させるためには，そのための考え方をプログラミング言語で記述して伝える必要がある．これから，プログラミング言語 Haskell[*1]を用いてプログラミングの重要な考え方を学んでいく．

プログラミングを学ぶときは，頭で理解するだけではなく，実際にプログラミングを行って作成したプログラムをコンピュータで動かすことが重要である．本テキストに出てくるプログラムは，実際に皆さんのコンピュータで動かして理解を深めてほしい．

1.1 GHCi の起動と終了

GNOME 端末上で，コマンド `ghci` により， Glasgow Haskell Compiler (GHC) [*2]の解釈器（インタープリタ）である GHCi を起動する：

```
$ ghci
GHCi, version 7.4.1: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude>
```

`Prelude>` は GHCi のプロンプトで，ここで Haskell の式などを入力して，評価することができる．

例えば，足し算の式を評価（簡約）する：

```
Prelude> 12+34
46
Prelude>
```

評価結果には，値を表す標準表現が返される．このような利用者とコンピュータの一連のやりとりを **セッション** という．

セッションを終えたいときには `:quit`，あるいは Ctrl-D を入力する：

[*1] 公式ページ (英語) `http://www.haskell.org`

[*2] `http://www.haskell.org/ghc/`

```
Prelude> :quit
Leaving GHCi.
$
```

問題 1 (B 問題). 次の式を GHCi で評価しなさい．評価結果が自分の考えた結果と同じか確認しなさい.

- `1+2`
- `1-2`
- `3*4`
- `3/4`
- `1+2*3`
- `(1+2)*3`

1.2 スクリプト

Haskell による定義をまとめて記述したものを**スクリプト**という．例えば，`sample.hs` というファイルに

sample.hs
```
x = 3 * 4
y = x - 2
double(x) = x + x
```

のように定義を並べたものがスクリプトである．この例では，1 行目は変数 `x`，2 行目は変数 `y`，3 行目は関数 `double` をそれぞれ定義している．Haskell のスクリプトのファイル名には，接尾辞`.hs` を用いるのが慣例である.

Haskell では行頭に空白を入れると意味が変わる．例えば,

```
x = 3 * 4
 y = x - 2
double(x) = x + x
```

のように 2 行目に空白があると構文エラーになる．後半に取り上げる C 言語では，行頭に空白が入っても意味が変わらない場合も多いので注意して欲しい.

カレントディレクトリにあるスクリプトをセッションで読み込むには`:load` **ファイル名** もしくは `:l` **ファイル名**と入力する.

```
Prelude> :l sample.hs
[1 of 1] Compiling Main             ( sample.hs, interpreted )
Ok, modules loaded: Main.
*Main>
```

スクリプトの構文が正しければ，“OK” と表示され，プロンプトが `*Main>` に変わる．スクリプトにエラーがあれば，“Failed” と表示される．例えば，上記のように行頭に空白を入れたスクリプトを読み込むとエラーになり次のように表示される.

```
Prelude> :l sample.hs
[1 of 1] Compiling Main             ( sample_error.hs, interpreted )

sample_error.hs:2:4: error:
    parse error on input ‘=’
    Perhaps you need a ’let’ in a ’do’ block?
    e.g. ’let x = 5’ instead of ’x = 5’
Failed, modules loaded: none.
Prelude>
```

エラーメッセージの最初の sample.hs:2:4 がエラーのあるファイル名，行数，その行での文字数なので（この場合は，sample.hs の 2 行目の 4 文字目)，その近辺をよく見てエラーを直そう．

一度読み込んだファイルを再度読み込むには:reload もしくは :r と入力すればよい．例えば，上記のエラーが出た後に，sample.hs を修正して :r と入力すると，修正されたスクリプトが読み込まれる．

エラーなく読み込まれた場合は，式を入力して評価することができる．関数ならば，実際に数値を指定して計算することができる．

```
*Main> double(3)
6
*Main>
```

関数を作成したら，実際に数値を指定して計算し，関数が間違っていないか必ず確認しよう．

問題 2 (B 問題)**.** sample.hs を作成して，セッションから読み込み，次の式を評価してみなさい．評価結果が自分の考えた結果と同じか確認しなさい．

- x
- y
- double(2)
- double(x)
- double(y)
- double(x+y+1)
- double(double(3))
- double(5) + x

1.3 コメント

Haskell のスクリプト中にコメントを記述することもできる．コメントはプログラムの評価時には無視されるので，日本語でメモを書いたり，関数の評価結果例を載せたりできる．Haskell のコメントには 2 種類ある．

- -- から行末までがコメント
- {- と -} に囲まれた部分がコメント．{- と -}は異なる行にあってもよい．

sample.hs

```
-- 最初の Haskell プログラム

{- 変数定義 -}
x = 3 * 4
y = x - 2

{-
x を 2 倍する関数
評価結果例:
*Main> double(3)
6
-}
double(x) = x + x
```

1.4 (参考) セッション

セッションの中で変数や関数を定義することもできる．定義を行うには `let` というキーワードを用いる：

```
Prelude> let x = 3 * 4
Prelude> let y = x - 2
```

定義された変数を用いて式を作ることができる：

```
Prelude> y - 1
9
```

関数を定義して使うこともできる：

```
Prelude> let double(x) = x + x
Prelude> double(y + 1)
22
```

ここで `double` の定義中に現れる `x` は，この定義においてのみ有効な変数であって，先ほど定義した `x` には影響がないことに注意してほしい．

問題 3 (A 問題)**.** 13 という値を持つ `thirteen` という変数をセッションの中で定義せよ．

解答例. 以下の通りである．

```
Prelude> let thirteen = 13
```

□

問題 4 (A 問題)**.** 与えられた数値を 4 倍する関数 `quadruple` をセッションの中で定義せよ．

解答例. 以下の通りである：

```
Prelude> let quadruple(x) = 4 * x
```

または，double を 2 回用いて合成関数にして

```
Prelude> let quadruple(x) = double(double(x))
```

とも表せる. □

問題 5 (A 問題)**.** 次の式を double と + の定義を用いて評価したときの簡約列を書きなさい．どの等式を用いたかを明示すること.

```
double(double(3))
```

解答例. 例えば，次のようになる.

```
  double(double(3))
= double(3 + 3)
= double(6)
= 6 + 6
= 12
```

□

2 式

数学で用いられたように Haskell においても，定数や変数，二項演算子といったものがある．定数には，数がある．例えば，3, -5, 99999999999 は定数である．変数名は，英小文字で始まり英文字，数字，もしくは_などの記号が続く識別子で表わされる．例えば，x, pi, x3, numOfSpace, num_of_space は変数名にできる．関数も同様の識別子で表される．例えば，f は関数を表す変数として用いられる．二項演算子は，記号のみから表される．例えば，二項演算子には，+, -, *といったものがある.

Haskell プログラムにおける式は，定数，変数，関数の式への適用，二項演算子による式の結合[*3]，式を括弧で括ったものなどである[*4]．例えば，fact(n) という式は，関数 fact を n という変数からなる式に適用している[*5]．注意してほしいのは，fact を n に適用しているのであって，n を fact に適用しているのではない．ここでは適用するものが関数であるから関数適用とも呼ぶ．また，1 + 3, x * y, x * f(n), (1 + 3)* 5 といったものも式である.

演算子には**優先順位**がある．例えば，関数適用，かけ算 (*)，足し算 (+) の順に優先される．関数の適用よりも演算子の適用のほうが順位が高いことに注意してほしい.

[*3] 厳密には，関数の式への適用，二項演算子による式の結合は式の式への適用の一種である．例えば，n + 1 は，+ という関数（中置演算子）を n という変数からなる式に適用してできた式 (n+) を 1 という定数からなる式に適用して得られた式である.

[*4] 他には，if 式や let 式などもあるが後述する.

[*5] 本授業では，関数適用は空白で表さない．すなわち，Haskell の記法では，fact n が一般的であるが，本授業では括弧を省略せずに fact(n) と記す.

問題 6 (A 問題). 3 は式であるか．変数 n は式であるか．n+3 は式であるか．

解答例. 以下の通り．

- 3 は定数である．定数は式である．したがって，3 は式である．
- n は変数である．変数は式である．したがって，n は式である．
- 3 と n は式である．2 つの式に + を適用したものも式である．したがって，n+3 は式である．

□

問題 7 (A 問題). 関数 fact を 3 に適用して式を作りなさい．また適用結果が式になる理由を説明しなさい．

解答例. 適用結果は，fact(3) である．前問の解答例より 3 は式である．関数を式に適用したものは式である．したがって fact(3) は式である． □

問題 8 (A 問題). 式 f(n) + 2 * y における演算子の優先順位を明示するために括弧をつけて記述しなさい．

解答例. (f(n)) + (2 * y) □

問題 9 (B 問題). f(3) + 2 * y は，なぜ式であるかを説明しなさい．ただし，+ と* は式を左右にひとつずつ取って式を計算する演算子（式）と見なして良い．

解答例. f(3) + 2 * y は，前問の解答例より括弧を省略せずに書くと (f(3)) + (2 * y) となる．f(3) は前々問の解答例より式である．2 は定数である．定数は式である．したがって，2 は式である．また，y は変数である．変数は式である．したがって，y は式である．二項演算子*が 2 つの式 2 および y を結合した 2 * y も式である．同様に二項演算子 + が 2 つの式 f(3) および 2 * y を結合した (f(3)) + (2 * y) も式である． □

2.1 値

式は値を**表す**（denote）．式の表す値を，その式の値という．

問題 10 (B 問題). 以下の式の値を手で導き出しなさい．導き出した値が GHCi で評価した値と一致することを確かめなさい．

- 12
- 1 + 2
- 3 * 4
- 1 + 2 * 3
- 3 - 1 - 1
- - 3

解答例. 以下の通り.

- 12
- 3
- 12
- 7
- 1
- -3

□

3 関数

Haskell では，前回学んだ関数の記述をほぼそのまま書き下すだけでコンピュータで値を与えて評価することができる.

例えば，2 倍する関数 $double(x) = x + x$ は，Haskell では次のようにして定義できる.

```
double(x) = x + x
```

問題 11 (B 問題)**.** 2 つの数の和を計算する関数 `add` を作成せよ.

解答例. 以下の通り.

```
add(x,y) = x + y
```

□

問題 12 (B 問題)**.** 与えられた数の 3 乗を計算する関数 `cube` を作成せよ.

解答例. 以下の通り.

```
cube(n) = n * n * n
```

□

問題 13 (B 問題)**.** 与えられた 2 つの数の 3 乗の和を計算する関数 `addcube` を作成せよ. 例えば，`addcube(2,3) = 35` となる. ただし，関数 `cube` と関数 `add` を用いよ.

解答例．以下の通り．

```
addcube(x,y) = add(cube(x), cube(y))
```

□

4 漸化式

階乗の計算は，

$$fact(0) = 1$$
$$fact(n) = n \times fact(n-1) \qquad \textbf{if } n \geq 1$$

と再帰的に定義されていた．Haskell では，この階乗の定義をほぼそのまま書き下すだけで階乗関数が実現できる[*6]：

```
fact(0) = 1
fact(n) = n * fact(n-1)
```

fact が関数名，= の左側の 0 と n が引数，右辺の式が本体である．式の中で関数が引数に適用されると，関数定義中の左辺が右辺に置き換えられる．このとき，引数に現れた変数は，具体的な式に置き換えられる．

例えば，関数 fact を定数 0 に適用した fact(0) という式を考える．この式は，1 行目の左辺に一致し，右辺に置き換えられ 1 というより簡単な式に変換（**簡約**）される．また，式 fact(5) を考える．この式は，2 行目の左辺に一致し，n が 5 となるように n が具体化され，右辺に置き換えられ，5 * fact(5 - 1) に簡約される．さらに，部分式 fact(4) も同じように簡約される．このようにして，

```
  fact(5)
= 5 * fact(5 - 1)
= 5 * fact(4)
= 5 * (4 * fact(4 - 1))
= 5 * (4 * fact(3))
= 5 * (4 * (3 * fact(3 - 1)))
= 5 * (4 * (3 * fact(2)))
= 5 * (4 * (3 * (2 * fact(2 - 1))))
= 5 * (4 * (3 * (2 * fact(1))))
= 5 * (4 * (3 * (2 * (1 * fact(0)))))
= 5 * (4 * (3 * (2 * (1 * 1))))
= 5 * (4 * (3 * (2 * 1)))
= 5 * (4 * (3 * 2))
= 5 * (4 * 6)
= 5 * 24
```

[*6] 以降，関数はスクリプトとしてファイルに記述するものとする

```
= 120
```

という簡約列が得られる．ここで，下線によって**簡約可能式**（redex）を示した．最後の式 120 の値は 120 であり，すなわち 5! に対応する．

問題 14 (A 問題)**.** `fact(3)` を評価せよ．`fact(42)` は評価できるであろうか.

解答例. セッションで評価を行った場合，以下の通りである：

```
*Main> fact(3)
6
*Main> fact(42)
1405006117752879898543142606244511569936384000000000
```

`fact(42)` を手で評価するのはとても大変である． □

問題 15 (A 問題)**.** 0 から与えられた非負整数（0 または正整数）までの整数の総和を計算するプログラム `mysum` [*7] を作成せよ．また，`mysum(3)` の簡約列を示しなさい.

解答. 例えば，関数定義は,

```
mysum(0) = 0
mysum(n) = n + mysum(n-1)
```

である．簡約列は

```
  mysum(3)
= 3 + mysum(3 - 1)
= 3 + mysum(2)
= 3 + (2 + mysum(2 - 1))
= 3 + (2 + mysum(1))
= 3 + (2 + (1 + mysum(0)))
= 3 + (2 + (1 + 0))
= 3 + (2 + 1)
= 3 + 3
= 6
```

となる． □

問題 16 (B 問題)**.** 次の式で定まる数列の第 n 項を求める関数 `c` を定義し，`c(100)` の値を求めよ.

$$c_0 = 1$$
$$c_n = 2 \times c_{n-1} + 1 \qquad \textbf{if } n \geq 1$$

[*7] Haskell では標準で `sum` という関数があらかじめ定義されているので，それと区別するために，ここでは `mysum` という関数とする．以降でも同様に標準関数と同じ関数を定義する場合には，関数名の最初に `my` を付けることにする.

解答例．解答例は以下の通り．

```
c(0) = 1
c(n) = 2 * c(n-1) + 1

c(100) = 2535301200456458802993406410751
```

□

問題 17 (B 問題)．0 から与えられた非負整数までの整数の 2 乗和を計算するプログラム sumSquare を作成せよ．また，sumSquare(5) の値は何であるか．

また，2 乗を求める関数 square を定義して，sumSquare と同じ計算をする sumSquare2 を定義しなさい．ただし，sumSquare2 の定義において，関数 square を利用すること．

解答例．解答例は以下の通り．

```
sumSquare(0) = 0
sumSquare(n) = n*n + sumSquare(n-1)
```

補助関数を利用した場合は以下の通り．

```
square(n) = n*n

sumSquare2(0) = 0
sumSquare2(n) = square(n) + sumSquare2(n-1)
```

□

問題 18 (B 問題)．前回資料の式 (15)–(17) で表されるフィボナッチ数列を Haskell で定義し，fib_{25} を求めなさい．

解答例．関数定義は

```
fib(0) = 0
fib(1) = 1
fib(n) = fib(n-1) + fib(n-2)
```

とすることができ，fib 25 の値は 75025 である． □

5 場合分け

Haskell の式には，if 式がある．if 式は 3 つの式 e_1, e_2, e_3 から，

```
if e1 then e2 else e3
```

のように構成される．ここで式 e_1 は論理値を表すものである．この if 式の値は，e_1 の値が真である場合，式 e_2 の値になり，e_1 の値が偽である場合，式 e_3 の値になる．

前回考えた絶対値を計算する関数を Haskell で `myabs` として定義するには，場合分けが必要である．

$$myabs(x) = \begin{cases} x & \textbf{if } x \geq 0 \\ -x & \textbf{otherwise} \end{cases} \tag{1}$$

例えば，`if` 式を用いると

```
myabs(x) = if x >= 0 then x else - x
```

と定義できる．式 `x >= 0` は論理値を表す．例えば，`x` が 3 のときに何が返されるかは，セッションで

```
Prelude> 3 >= 0
True
```

と試すことで確認できる．

問題 19 (A 問題)**.** 次の if 式

```
if x >= 0 then x else - x
```

が確かに式であることを説明せよ．

解答例. `x >= 0`, `x - x` は式である．上述のように if 式は 3 つの式から構成されるので，与えられたものは確かに式である． □

then や else の後の式を if 式として，複数の if 式を組み合わせることもできる．例えば，$|x| + |x - 2|$ の値をもつ式は，$x > 2$, $x < 0$, およびそれ以外（$0 \leq x \leq 2$）に場合分けを行うと，

```
if x > 2 then x + x - 2
         else (if x < 0 then (-x) + (- (x - 2))
                        else x + (- (x - 2)))
```

と表される．この式は，算術計算を行って括弧を省略すると，

```
if x > 2 then 2 * x - 2
         else if x < 0 then 2 - 2 * x
                       else - 2
```

とも表される．

問題 20 (A 問題)**.** 入力が正であったら 1，0 であったら 0，負であったら -1 を出力する関数 *sign* を Haskell で定義せよ．

$$sign(x) = \begin{cases} 1 & \textbf{if } x > 0 \\ 0 & \textbf{if } x = 0 \\ -1 & \textbf{otherwise} \end{cases}$$

解答例. 以下の通りである.

```
sign(x) = if x > 0 then 1
          else if x == 0 then 0
          else -1
```

なお，Haskell での等価演算子は == であることに注意せよ．x と 0 が等しいかどうかは，x = 0 ではなく，x == 0 である． □

Haskell の等価判定，大小比較は表 1 の通りである.

表 1　等価判定，大小比較

意味 (True となるとき)	Haskell での記述	(参考) C での記述
A と B が等しい ($A = B$)	`A == B`	`A == B`
A と B が等しくない ($A \neq B$)	`A /= B`	`A != B`
A は B より大きい ($A > B$)	`A > B`	`A > B`
A は B 以上 ($A \geq B$)	`A >= B`	`A >= B`
A は B 未満 ($A < B$)	`A < B`	`A < B`
A は B 以下 ($A \leq B$)	`A <= B`	`A <= B`

問題 21 (B 問題)**.** 2 つの数の差 (大きい数から小さい数を引いた値) を計算する関数 `diff` を作成せよ.

解答例. 以下の通り.

```
diff(x,y) = if x > y then x - y else y - x
```

□

問題 22 (A 問題)**.** 2 つの数 x, y のうち小さい方を計算する関数 `min2(x,y)` を作成せよ.

解答例. 以下の通り.

```
min2(x,y) = if x >= y then y else x
```

□

問題 23 (B 問題)**.** 3 つの数のうち最も小さいものを計算する関数 `min3` を，`min2` を 2 つ用いて定義せよ.

解答例. 以下の通り.

```
min3(x,y,z) = min2(x,min2(y,z))
```

もしくは

```
min3(x,y,z) = min2(min2(x,y),z)
```

□

問題 24 (B 問題). 2 つの数のうち大きい方を計算する関数 max2 を作成せよ.

解答例. 以下の通り.

```
max2(x,y) = if x >= y then x else y
```

□

問題 25 (B 問題). 3 つの数のうち最も大きいものを計算する関数 max3 を，max2 を 2 つ用いて定義せよ.

解答例. 以下の通り.

```
max3(x,y,z) = max2(x,max2(y,z))
```

もしくは

```
max3(x,y,z) = max2(max2(x,y),z)
```

□

問題 26 (B 問題). 3 つの数のうち最も大きいものを計算する関数 max3if を，関数 max2 を使わずに定義せよ．つまり，if の組み合わせで作成せよ.

解答例. 以下の通り.

```
max3if(x,y,z) = if x>y then
                    if z>x then z
                    else x
                else
                    if z>y then z
                    else y
```

□

問題 27 (B 問題)．4 つの数のうち最も大きいものを計算する関数 max4 を作成せよ．関数 max2 を使ってもよいし，使わなくてもよい．

解答例．以下の通り．

```
max4(w,x,y,z) = max2(max2(w,x),max2(y,z))
```

□

問題 28 (D 問題)．3 つの整数値の中央値を計算する関数 median を作成せよ．ヒント：以下の式を評価するとすべて 2 になれば良い．

```
median(1,2,3)
median(1,3,2)
median(2,1,3)
median(2,3,1)
median(3,1,2)
median(3,2,1)
```

解答例．以下の通り．

```
median(x,y,z) = if x >= y then
                  if y >= z then y
                            else if x >= z then z
                                           else x
                else
                  if x >= z then x
                            else if y >= z then z
                                           else y
```

□

6 章末問題

問題 29 (B 問題)．前回資料の問題 24 の *square* を Haskell で定義せよ．

解答例．以下の通り．

```
square(x) = x*x
```

□

問題 30 (B 問題)．前回資料の問題 25 の *plus1* を Haskell で定義せよ．

解答例．以下の通り．

```
plus1(x) = x+1
```

□

問題 31 (B 問題)．前回資料の問題 26 の *sqplus1* を Haskell で定義せよ．

解答例．以下の通り．

```
sqplus1(x) = plus1(square(x))
```

□

問題 32 (B 問題)．前回資料の問題 27 のリュカ数列を Haskell で関数 `lucas` として定義せよ．つまり，`lucas(0)` は L_0 を，`lucas(n)` は L_n を計算する．

解答例．以下の通り．

```
lucas(0) = 2
lucas(1) = 1
lucas(n) = lucas(n-1) + lucas(n-2)
```

□

問題 33 (B 問題)．前回資料の問題 28 のトリボナッチ数列を Haskell で関数 `tribo` として定義せよ．つまり，`tribo(0)` は T_0 を，`tribo(n)` は T_n を計算する．

解答例．以下の通り．

```
tribo(0) = 0
tribo(1) = 0
tribo(2) = 1
tribo(n) = tribo(n-1) + tribo(n-2) + tribo(n-3)
```

□

問題 34 (B 問題)．前回資料の問題 29 の 1 から n までの整数の 2 乗の和を求める数列 $\{a_n\}_{n\geq 1}$ を Haskell の関数 `sumsq` として定義せよ．つまり，`sumsq(1)` は a_1 を，`sumsq(n)` は a_n を計算する．

解答例. 以下の通り.

```
sumsq(1) = 1
sumsq(n) = n^2 + sumsq(n-1)
```

□

問題 35 (B 問題). 前回資料の問題 30 の 1 から n までの整数の 3 乗の和を求める数列 $\{a_n\}_{n\geq 1}$ を Haskell の関数 `sumcb` として定義せよ. つまり, `sumcb(1)` は a_1 を, `sumcb(n)` は a_n を計算する.

解答例. 以下の通り.

```
sumcb(1) = 1
sumcb(n) = n^3 + sumcb(n-1)
```

□

問題 36 (B 問題). 前回資料の問題 31 の 2 から n 番目までの偶数の 2 乗の和を求める数列 $\{a_n\}_{n\geq 1}$ を Haskell の関数 `sumsqeven` として定義せよ. つまり, `sumsqeven(1)` は a_1 を, `sumsqeven(n)` は a_n を計算する.

解答例. 以下の通り.

```
sumsqeven(1) = 4
sumsqeven(n) = (2*n)^2 + sumsqeven(n-1)
```

□

問題 37 (B 問題). 前回資料の問題 32 の 1 から n 番目までの奇数の 2 乗の和を求める数列 $\{a_n\}_{n\geq 1}$ を Haskell の関数 `sumsqodd` として定義せよ. つまり, `sumsqodd(1)` は a_1 を, `sumsqodd(n)` は a_n を計算する.

解答例. 以下の通り.

```
sumsqodd(1) = 1
sumsqodd(n) = (2*n-1)^2 + sumsqodd(n-1)
```

□

問題 38 (B 問題). 前回資料の問題 33 を参考にして, x の n 乗 (x^n) を計算する Haskell の関数 `pow(x, n)` を定義しなさい. なお, `n>=0` としてよい.

解答例. 以下の通り.

```
pow(x, 0) = 1
pow(x, n) = x * pow(x, n-1)
```

□

問題 39 (D 問題). 前回資料の問題 34 の n 勝先取の勝負で，どの 1 試合も A と B のそれぞれが勝つ確率が p と $1-p$ のとき，A が i 勝 j 敗のときの優勝確率 $A_{i,j}$ は次のように定義できる.

$$
\begin{aligned}
A_{n,j} &= 1 \\
A_{i,n} &= 0 \\
A_{i,j} &= p \times A_{i+1,j} + (1-p) \times A_{i,j+1} \qquad \textbf{if } i < n \wedge j < n
\end{aligned}
$$

これを Haskell の関数 `winA(n, p, i, j)` として定義しなさい. なお，`n>0`, `p<=1`, `i<=n`, `j<=n` としてよい.

解答例. 以下の通り.

```
winA(n, p, i, j) = if i==n then 1
                   else if j==n then 0
                   else p * winA(n, p, i+1, j) + (1-p)* winA(n, p, i, j+1)
```

□

問題 40 (D 問題). プロ野球の日本シリーズやクライマックスシリーズのファイナルステージは 4 勝先取の勝負である. 問題 39 で定義した関数を利用し，4 勝先取の勝負における次の優勝確率を求めなさい.

1. A の勝つ確率が 0.5 で，A が 0 勝 0 敗のときの優勝確率
2. A の勝つ確率が 0.5 で，A が 3 勝 0 敗のときの優勝確率
3. A の勝つ確率が 0.5 で，A が 0 勝 3 敗のときの優勝確率
4. A の勝つ確率が 0.6 で，A が 0 勝 0 敗のときの優勝確率
5. A の勝つ確率が 0.7 で，A が 1 勝 0 敗のときの優勝確率

解答例. 以下の通り.

```
winA(4, 0.5, 0, 0) = 0.5
winA(4, 0.5, 3, 0) = 0.935
winA(4, 0.5, 0, 3) = 6.25e-2
winA(4, 0.6, 0, 0) = 0.710208
winA(4, 0.7, 1, 0) = 0.92953
```

□

問題 41 (D 問題). 整数 k 以上から整数 n 以下の整数の総和を計算するプログラム sumS(k,n) を作成せよ. プログラム中では，再帰を使い，一回の再帰ごとに求める範囲が狭くなるように実現しなさい．なお，$0 \leq$ k $\leq$ n のとき，sumS(k,n) の値は $\sum_{i=k}^{n} i$ の値に対応する.

解答. 関数定義の一例は

```
sumS(k,n) = if k > n then 0
                     else k + sumS(k+1,n)
```

である. □