プログラミング言語**特**論（2007年度）・テスト問題用紙

('08年2月14日（木）・ 13:00 〜 14:30)

解答上、その他の注意事項

I. 問題は、問I〜Vまである。

II. 解答用紙の右上の欄に学籍番号・名前を記入すること。

III. ノート・プリント・参考書などは持ち込み可である。

IV. 携帯電話などの通信機能を持つものは持ち込み不可である。

V. 問Iを解答するときのみ、ノートPCを使用して良い。ネットワークに接続してWWWを閲覧しても良いが、掲示板、チャット、メールなどで生身の人間と通信することは禁じる。

VI. テストの配点は50点（+ボーナス20点）である。合格はレポートの得点を加点して、100点満点中60点以上とする。

I. （Haskell 実習問題） （6 点 × 2）

(1) 引数として与えられる整数のリスト中の負の要素の個数を返す関数

```
foo :: [Integer] -> Integer
```

を定義せよ。

例えば、`foo [2,-2,6,-2]` は 2、`foo [4,8,-1,0,-4,-3]` は 3 となる。

この問では `map`, `filter` などのライブラリ関数や内包表記を使わず、`if`〜`then`〜`else`〜式や論理演算子、不等号、パターンマッチ、再帰などのみを使って定義せよ。

(2) 引数として与えられる整数のリストの中で、昇順に隣り合う 3 つの要素の組（つまり、$y == x + 1$ かつ $z == y + 1$ がなりたつ 3 つの要素の組 (x, y, z)）を列挙する関数

```
bar :: [Integer] -> [(Integer,Integer,Integer)]
```

を（リストの内包表記を用いて）定義せよ。

例えば、`bar [2,5,4,6,3]` は `[(2,3,4),(3,4,5),(4,5,6)]`、`bar [2,9,6,10,3]` は `[]` となる。（リストの要素の順番はこのとおりでなくても良い。）

II. （ラムダ計算）（6 点 × 2）

次の λ 式が正規形に到達するまでの、最左変換による 1 ステップずつの β 変換の列を書け。ただし、正規形が存在しない式については、それが判別できる時点（ただし少なくとも 3 回以上 β 変換したあと）に「停止しない」と記入せよ。

記入例:

$$
\begin{aligned}
& (\lambda fx.f(fx))((\lambda fx.f(fx))g)y \\
\xrightarrow{\beta}\ & (\lambda x.((\lambda fx.f(fx))g)(((\lambda fx.f(fx))g)x))y \\
\xrightarrow{\beta}\ & ((\lambda fx.f(fx))g)(((\lambda fx.f(fx))g)y) \\
\xrightarrow{\beta}\ & (\lambda x.g(gx))(((\lambda fx.f(fx))g)y) \\
\xrightarrow{\beta}\ & g(g(((\lambda fx.f(fx))g)y)) \\
\xrightarrow{\beta}\ & g(g((\lambda x.g(gx))y)) \\
\xrightarrow{\beta}\ & g(g(g(gy)))
\end{aligned}
$$

$$
\begin{aligned}
& (\lambda x.xx)(\lambda x.xx) \\
\xrightarrow{\beta}\ & (\lambda x.xx)(\lambda x.xx) \\
\xrightarrow{\beta}\ & (\lambda x.xx)(\lambda x.xx) \\
\xrightarrow{\beta}\ & (\lambda x.xx)(\lambda x.xx) \\
\xrightarrow{\beta}\ & \text{停止しない}
\end{aligned}
$$

(1) $(\lambda xy.yxy)(\lambda zw.z)(\lambda xy.y)$

(2) $(\lambda mnf.m(nf))(\lambda sz.s(sz))(\lambda sz.s(s(sz)))$

なお、必要に応じて $I \equiv \lambda x.x$ など適宜、定数を定義しても良い。

III. （Haskell）（7 点 ×2）

次の例にならって、下の Haskell のプログラム (1)〜(2) を評価した結果を書け。

例: `take 5 (from 1)` ⇒ **答:** `[1,2,3,4,5]`

ただし、`take` と `from` は講義プリントに定義されているとおりの関数である。

```
from :: Integer -> [Integer];
from n = n : from (n+1);

take :: Integer -> [a] -> [a];
take 0 _      = [];
take _ []     = [] ;
take n (x:xs) = x : take (n-1) xs;
```

(1) `foldl (\ x y -> 2*x + y) 0 [1,2,3]`

この問で使用されている関数 `foldl` の定義は次のとおりである。

```
foldl :: (a -> b -> a) -> a -> [b] -> a;
foldl f z []     = z;
foldl f z (x:xs) = foldl f (f z x) xs;
```

(2) `[ y-x | x <- [1,2,3], y <- [2,4,6], x <= y]`

（この問に関してはリスト内の順番の間違いの減点は 1 点のみとする。）

IV. （語句） （6 点 × 2）

プログラミング言語（やその処理系）で用いられる次の 6 つの語句のうち 2 つを選択し、具体的な例を挙げて説明せよ。ただし、講義プリントにのっている例ではなく、オリジナル の例を考えること。

- 抽象構文（abstract syntax）
- 動的束縛（dynamic binding）
- 高階関数（higher-order function）
- 例外（exception）
- 非決定性（nondeterminism）
- 接続（continuation）

V. （自由記述 — ボーナス問題） （最高 20 点）

これまでにいろいろなプログラミング言語を学習して、理解しにくかった点、間違いやすいと感じた点、あるいは初心者がつまづきやすいと感じた点は何か? できるだけ具体的に述べよ。また、それに対して、オリジナルの改良のアイデアや抜本的な解決策を（実現性は気にせず）自由に考述せよ。

プログラミング言語**特**論（2007年度）・テスト解答用紙('08年2月14日)

学籍番号		氏名	

学籍番号		氏名	