『プログラム言語論』 期末試験 解答例
2014 年 6 月 26 日 (木)

問 1. (配点 30 点) MiniC 言語で書かれた次のプログラムについて以下の問に答えよ.

```
int z;
int f (int x) {
  print x;              /* C 言語の printf("%d\n", x); と同じ*/
  return x;
}
int g (int x, int y) {
  z = 40;
  return x * x * x;
}
int main () {
  int z;
  z = 30;
  print g(f(10),f(20));
  print z;
}
```

上記のプログラムを以下のそれぞれの方式で実行したとき、印刷される値を、印刷される順番に示しなさい. (試験中に補足: 1-a, 1-d において複数の引数を持つ関数においては、引数を左から右に評価するものと仮定しなさい。)

1-a. 静的束縛かつ値呼び: **答** `10 20 1000 30`

1-b. 静的束縛かつ名前呼び **答** `10 10 10 1000 30`

1-c. 静的束縛かつ必要呼び **答** `10 1000 30`

1-d. 動的束縛かつ値呼び **答** `10 20 1000 40`

補足 静的束縛の 3 つのケースは特に解説の必要はないと思う. 最後の動的束縛のケースは, z=40; で代入される z が, グローバルに (1 行目において) 宣言された z ではなく main の中でローカルに (11 行目において) 宣言された z であるというのがポイントである.

MiniC 言語における f (e) という形の関数呼出し (関数 f を実引数 e に適用) に対して、以下の文がそれぞれ正しいかどうか判定し、その根拠を簡潔に (1-2 行で) 書きなさい。

1-e. 関数 f の本体で引数が使われていないとき、名前呼び戦略と必要呼び戦略においては、e の計算を行わずに済むため、値呼び戦略より有利である。

答 正しい. 値呼び戦略では, e の計算が 1 回行れるが, 名前呼びおよび必要呼び戦略では, e の計算はその値が必要とされるときのみ行われるので, この場合は計算されない.

1-f. 関数 f の本体で引数が 2 回以上使われるとき、名前呼び戦略と必要呼び戦略においては、e の計算を 2 回以上行うため、値呼び戦略より不利である。

答 正しくない．必要呼び戦略では，e の計算はその値が必要とされるときに 1 回だけ行われ，2 回目以降に必要とされるときはその値を利用する (計算は行わない) ので，e の計算は 1 回のみである．

補足 ここでは,「引数の計算が何回行われるか」という観点のみで比較をしたが，実装にあたっては，値呼びより名前呼びや必要呼びの方が複雑な仕組みを要するため (計算結果ではなく，計算する前の式を保存しておく必要があるため)，実際の計算の効率は一概に比較できない．その点に基づいて比較した場合，たとえば「1-e」の答えは,「正しくない」ということになる．それでも (その根拠が正しく書いてあれば) 正解である．なお，本問においては「根拠を簡潔に書きなさい」と明示してあるので，答えだけが書いてあるものは 0 点である．

問 2. (配点 30 点、加点問題あり) 次の抽象構文で定義される言語 L に対する抽象機械について考える。

$$e ::= \texttt{Var}(x) \mid \texttt{Int}(n) \mid \texttt{Plus}(e, e) \mid \texttt{Let}(x, e, e) \mid \texttt{Lam}(x, e) \mid \texttt{App}(e, e)$$

ただし、 x は変数を表し、n は整数定数 (-1 や 3 など) を表し、また、授業で定義した言語から $\texttt{Times}(e, e)$ を除いた。

この抽象機械は CEK 機械の一種であり、状態は、e (初期状態)、$\texttt{eval}\langle e \mid E \mid K\rangle$ の形 (eval 状態)、$\texttt{apply}\langle K \mid E\rangle$ の形 (apply 状態)、v (最終状態) のいずれかである。この状態の間の遷移は以下の規則で与えられる。

$$\begin{aligned}
e &\to \texttt{eval}\langle e \mid \texttt{[]} \mid \texttt{init}\rangle \\
\texttt{eval}\langle \texttt{Var}(x) \mid E \mid K\rangle &\to \texttt{apply}\langle K \mid v\rangle \quad (v = \texttt{lookup}(x, E)) \\
\texttt{eval}\langle \texttt{Int}(n) \mid E \mid K\rangle &\to \texttt{apply}\langle K \mid n\rangle \\
\texttt{eval}\langle \texttt{Plus}(e, e') \mid E \mid K\rangle &\to \texttt{eval}\langle e \mid E \mid \mathit{plus}1(e', E)\texttt{::}K\rangle \\
\texttt{eval}\langle \texttt{Let}(x, e, e') \mid E \mid K\rangle &\to \texttt{eval}\langle e \mid E \mid \mathit{letin}(x, e', E)\texttt{::}K\rangle \\
\texttt{eval}\langle \texttt{Lam}(x, e) \mid E \mid K\rangle &\to \texttt{apply}\langle K \mid \texttt{Closure}(x, e, E)\rangle \\
\texttt{eval}\langle \texttt{App}(e, e') \mid E \mid K\rangle &\to \texttt{eval}\langle e \mid E \mid \mathit{apply}1(e', E)\texttt{::}K\rangle \\
\texttt{apply}\langle \mathit{plus}1(e, E)\texttt{::}K \mid v\rangle &\to \texttt{eval}\langle e \mid E \mid \mathit{plus}2(v)\texttt{::}K\rangle \\
\texttt{apply}\langle \mathit{plus}2(n_1)\texttt{::}K \mid n_2\rangle &\to \texttt{apply}\langle K \mid n\rangle \quad (n_1 \dot{+} n_2 = n) \\
\texttt{apply}\langle \mathit{letin}(x, e, E)\texttt{::}K \mid v\rangle &\to \texttt{eval}\langle e \mid E[x = v] \mid K\rangle \\
\texttt{apply}\langle \mathit{apply}1(e, E)\texttt{::}K \mid v\rangle &\to \texttt{eval}\langle e \mid E \mid \mathit{apply}2(v)\texttt{::}K\rangle \\
\texttt{apply}\langle \mathit{apply}2(\texttt{Closure}(x, e, E))\texttt{::}K \mid v\rangle &\to \texttt{eval}\langle e \mid E[x = v] \mid K\rangle \\
\texttt{apply}\langle \texttt{init} \mid v\rangle &\to v \quad \text{(最終状態)}
\end{aligned}$$

ここで、$\texttt{[]}$,$\texttt{lookup}(x, E)$, $E[x = v]$ は環境 E に対する操作、$\dot{+}$ は加算、$X\texttt{::}K$ はスタック K に対する操作、値 v は、整数 n か、$\texttt{Closure}(x, e, E)$ の形をした式であり、すべて授業で与えた通りである。

このとき，以下のプログラムを初期状態として、最終状態に至るまでの状態遷移列を書きなさい．

2-a. $\texttt{Plus}(\texttt{Int}(10), \texttt{Int}(20))$

2-b. $\texttt{Let}(x, \texttt{Int}(10), \texttt{Plus}(\texttt{Var}(x), \texttt{Int}(20)))$

2-c. $\texttt{App}(\texttt{Lam}(x, \texttt{App}(\texttt{Var}(x), \texttt{Int}(10))), \texttt{Lam}(y, \texttt{Plus}(\texttt{Var}(y), \texttt{Int}(20))))$

答 以下の通りである．

$$
\begin{aligned}
\text{(2-a)}\ \ \texttt{Plus}(\texttt{Int}(10), \texttt{Int}(20)) &\to \texttt{eval}\langle \texttt{Plus}(\texttt{Int}(10), \texttt{Int}(20)) \mid \texttt{[]} \mid \texttt{init} \rangle \\
&\to \texttt{eval}\langle \texttt{Int}(10) \mid \texttt{[]} \mid \mathit{plus}1(\texttt{Int}(20), \texttt{[]})\texttt{::init} \rangle \\
&\to \texttt{apply}\langle \mathit{plus}1(\texttt{Int}(20), \texttt{[]})\texttt{::init} \mid 10 \rangle \\
&\to \texttt{eval}\langle \texttt{Int}(20) \mid \texttt{[]} \mid \mathit{plus}2(10)\texttt{::init} \rangle \\
&\to \texttt{apply}\langle \mathit{plus}2(10)\texttt{::init} \mid 20 \rangle \\
&\to \texttt{apply}\langle \texttt{init} \mid 30 \rangle \\
&\to 30
\end{aligned}
$$

$$
\begin{aligned}
\text{(2-b)}\ \ \texttt{Let}(x, \texttt{Int}(10), \texttt{Plus}(\texttt{Var}(x), \texttt{Int}(20))) &\to \texttt{eval}\langle \texttt{Let}(x, \texttt{Int}(10), \texttt{Plus}(\texttt{Var}(x), \texttt{Int}(20))) \mid \texttt{[]} \mid \texttt{init} \rangle \\
&\to \texttt{eval}\langle \texttt{Int}(10) \mid \texttt{[]} \mid \mathit{letin}(x, \texttt{Plus}(\texttt{Var}(x), \texttt{Int}(20)), \texttt{[]})\texttt{::init} \rangle \\
&\to \texttt{apply}\langle \mathit{letin}(x, \texttt{Plus}(\texttt{Var}(x), \texttt{Int}(20)), \texttt{[]})\texttt{::init} \mid 10 \rangle \\
&\to \texttt{eval}\langle \texttt{Plus}(\texttt{Var}(x), \texttt{Int}(20)) \mid \texttt{[]}[x = 10] \mid \texttt{init} \rangle \\
&\to \texttt{eval}\langle \texttt{Var}(x) \mid \texttt{[]}[x = 10] \mid \mathit{plus}1(\texttt{Int}(20), \texttt{[]}[x = 10])\texttt{::init} \rangle \\
&\to \texttt{apply}\langle \mathit{plus}1(\texttt{Int}(20), \texttt{[]}[x = 10])\texttt{::init} \mid 10 \rangle \\
&\to \texttt{eval}\langle \texttt{Int}(20) \mid \texttt{[]}[x = 10] \mid \mathit{plus}2(10)\texttt{::init} \rangle \\
&\to \texttt{apply}\langle \mathit{plus}2(10)\texttt{::init} \mid 20 \rangle \\
&\to \texttt{apply}\langle \texttt{init} \mid 30 \rangle \\
&\to 30
\end{aligned}
$$

(2-c) については、以下のように置く。

$$
\begin{aligned}
C_1 &= \texttt{Closure}(x, \texttt{App}(\texttt{Var}(x), \texttt{Int}(10)), \texttt{[]}) \\
C_2 &= \texttt{Closure}(y, \texttt{Plus}(\texttt{Var}(y), \texttt{Int}(20)), \texttt{[]})
\end{aligned}
$$

(2-c) $\texttt{App}(\texttt{Lam}(x, \texttt{App}(\texttt{Var}(x), \texttt{Int}(10))), \texttt{Lam}(y, \texttt{Plus}(\texttt{Var}(y), \texttt{Int}(20))))$

$\rightarrow \texttt{eval}\langle \texttt{App}(\texttt{Lam}(x, \texttt{App}(\texttt{Var}(x), \texttt{Int}(10))), \texttt{Lam}(y, \texttt{Plus}(\texttt{Var}(y), \texttt{Int}(20)))) \mid \texttt{[]} \mid \texttt{init}\rangle$

$\rightarrow \texttt{eval}\langle \texttt{Lam}(x, \texttt{App}(\texttt{Var}(x), \texttt{Int}(10))) \mid \texttt{[]} \mid \mathit{apply}1(\texttt{Lam}(y, \texttt{Plus}(\texttt{Var}(y), \texttt{Int}(20))), \texttt{[]})\texttt{::init}\rangle$

$\rightarrow \texttt{apply}\langle \mathit{apply}1(\texttt{Lam}(y, \texttt{Plus}(\texttt{Var}(y), \texttt{Int}(20))), \texttt{[]})\texttt{::init} \mid C_1\rangle$

$\rightarrow \texttt{eval}\langle \texttt{Lam}(y, \texttt{Plus}(\texttt{Var}(y), \texttt{Int}(20))) \mid \texttt{[]} \mid \mathit{apply}2(C_1)\texttt{::init}\rangle$

$\rightarrow \texttt{apply}\langle \mathit{apply}2(C_1)\texttt{::init} \mid C_2\rangle$

$\rightarrow \texttt{eval}\langle \texttt{App}(\texttt{Var}(x), \texttt{Int}(10)) \mid \texttt{[]}[x = C_2] \mid \texttt{init}\rangle$

$\rightarrow \texttt{eval}\langle \texttt{Var}(x) \mid \texttt{[]}[x = C_2] \mid \mathit{apply}1(\texttt{Int}(10), \texttt{[]}[x = C_2])\texttt{::init}\rangle$

$\rightarrow \texttt{apply}\langle \mathit{apply}1(\texttt{Int}(10), \texttt{[]}[x = C_2])\texttt{::init} \mid C_2\rangle$

$\rightarrow \texttt{eval}\langle \texttt{Int}(10) \mid \texttt{[]}[x = C_2] \mid \mathit{apply}2(C_2)\texttt{::init}\rangle$

$\rightarrow \texttt{apply}\langle \mathit{apply}2(C_2)\texttt{::init} \mid 10\rangle$

$\rightarrow \texttt{eval}\langle \texttt{Plus}(\texttt{Var}(y), \texttt{Int}(20)) \mid \texttt{[]}[y = 10] \mid \texttt{init}\rangle$

$\rightarrow \texttt{eval}\langle \texttt{Var}(y) \mid \texttt{[]}[y = 10] \mid \mathit{plus}1(\texttt{Int}(20), \texttt{[]}[y = 10])\texttt{::init}\rangle$

$\rightarrow \texttt{apply}\langle \mathit{plus}1(\texttt{Int}(20), \texttt{[]}[y = 10])\texttt{::init} \mid 10\rangle$

$\rightarrow \texttt{eval}\langle \texttt{Int}(20) \mid \texttt{[]}[y = 10] \mid \mathit{plus}2(10)\texttt{::init}\rangle$

$\rightarrow \texttt{apply}\langle \mathit{plus}2(10)\texttt{::init} \mid 20\rangle$

$\rightarrow \texttt{apply}\langle \texttt{init} \mid 30\rangle$

$\rightarrow 30$

また、次の問題に答えなさい。

2-d. $\texttt{Closure}(x, e, E)$ は関数クロージャと呼ばれる。関数クロージャを必要とする (それがないと、うまく実行できない) プログラムの例を 1 つあげなさい。なお、このプログラムに対する計算 (状態遷移列) を書く必要はない。

答の前に補足 関数クロージャは、実行時における「関数の値」であるが、単に関数の引数と本体をセットにしたものではなく、評価時点での環境を取り込んでいる点に特徴がある。これが必要なのは、静的束縛の関数型言語 (「関数」を値として取り扱える言語) でである。

答 プログラム例はいろいろあるが、一例は、以下の通りである.

$$\texttt{Let}(x, \texttt{Int}(10), \texttt{Let}(f, \texttt{Lam}(y, \texttt{Var}(x)), \texttt{Let}(x, \texttt{Int}(20), \texttt{App}(\texttt{Var}(f), \texttt{Int}(30)))))$$

なお、ここでは、この問題の言語 L の抽象構文にあわせて書いたが、特に言語を指定した問題ではないので、対応するとおもわれる具体構文で書いても, あるいは ML など他の言語で書いても、同じ趣旨のプログラムであれば正解とした。たとえば、以下のものである。

```
let x = 10 in
let f = fun y -> x in
let x = 20 in
 f 30
```

この例では、関数クロージャを作らなかったら (環境を取り込まずに、単に `fun y->x` という式を使ったら)、20 が返ってきてしまうが、関数クロージャとして `Closure(y,Var(x),[][x=10])` を作ることにより、正しく 10 を返すことができる。

一方で、「高階関数」とだけ書いたものは、「プログラム例を書け」という質問への答えになっていない。また、C や MiniC 言語は関数クロージャの処理が必要ないので、この問題の答えにはならない。

2-e. 上の抽象機械は、値呼び、名前呼び、必要呼びのどの方式に対応したものであるか、簡潔な根拠とともに述べよ。

答 値呼びである。なぜなら、関数呼び出し $\texttt{App}(e_1, e_2)$ の処理において、まず e_1 を計算してスタックに apply1(...) を積み、それが終わると今度はスタックに apply2(...) を積み、それが終わったあとに (遷移規則の最後の 2 番目の apply2(Closure...) のケースで)、関数の引数に値を束縛して、関数本体の計算をしているからである。これは、関数本体を呼び出す前に引数の計算を終えるので、値呼びである。(また、環境 E において、変数に束縛されるのが、値 (数かクロージャ) のみであることからも、値呼びであることがわかる。)

補足 答案は、上のように丁寧に書かなくても、関数呼び出しにおいて引数が計算されてから本体が呼ばれる、ということが書いてあれば正解とした。

2-f. (加点問題) 上の言語に、「引数 e に 1 を足す」ことを意味する $\texttt{Inc}(e)$ という式を加えたい。たとえば、$\texttt{Inc}(\texttt{Int}(17))$ の実行が 18 という最終状態で終わるようにしたい。このために、どのような状態遷移を追加すればよいか、示しなさい。

答 解答の一例は以下の通りである.

$$\texttt{eval}\langle \texttt{Inc}(e) \mid E \mid K\rangle \to \texttt{eval}\langle e \mid E \mid inc1::K\rangle$$
$$\texttt{apply}\langle inc1::K \mid n\rangle \to \texttt{apply}\langle K \mid n'\rangle \quad (n \dot{+} 1 \;=\; n')$$

補足 問題の趣旨としては、上記解答のように答えてもらうつもりだったのだが、この問題には以下のようなもっと簡単な別解もある。

$$\texttt{eval}\langle \texttt{Inc}(e) \mid E \mid K\rangle \to \texttt{eval}\langle \texttt{Plus}(e, \texttt{Int}(1)) \mid E \mid K\rangle$$

あるいは

$$\texttt{eval}\langle \texttt{Inc}(e) \mid E \mid K\rangle \to \texttt{eval}\langle e \mid E \mid plus2(1)::K\rangle$$

問 3. (配点 40 点、加点問題あり)

以下の 4 つの事項の**すべて**について、それぞれ 5 行程度で説明しなさい。

3-a. MiniC 言語 (あるいは C, Java, ML などの言語) において、以下の情報が静的に決定できるかどうか答えた上で、プログラムにおける「静的情報」とは何かを説明しなさい。(i) 関数の実行において、その引数の値が実際に 1 回以上使われるかどうか。(ii) 整数に対する足し算の実行において、その引数が実際に整数であるかどうか。

答 (一例) 静的情報 (静的に決定できる情報) は，プログラムの字面のみから (プログラムを実行せずに) 決定できる情報のことである．(i) は静的には決定できない．(たとえば，`if e {y=x;} else 0;` という MiniC プログラムでは，e が 0 かどうかによって，x が使われるかどうかがかわる．しかし任意の式 e の値が 0 になるかどうかは静的には決められない．) (ii) は静的に決定できる．(MiniC 言語は C 言語のサブセットと考えられ，C 言語は静的に型検査を行なうことによって，足し算の引数が整数であるかどうかを判定できる．)

3-b. ヒープとは何か、また、プログラム言語の処理系がヒープを必要とする具体的な理由は何か、そして、ヒープの適切な管理のためにプログラム言語の処理系が必要とする機能は何か、について、説明しなさい。

答 (一例) ヒープは，プログラムが実行時に使うメモリ領域のうち，ブロック構造の処理に必要なスタック以外の部分を指し，永続的なデータ (関数呼出しを抜けた後でも利用することがあるデータ) を格納するために使う．永続的なデータの例としては対など構造をもつデータ，関数クロージャ，文字列，(オブジェクト指向言語の) オブジェクトなどがある．Lisp や Java ではヒープは言語処理系が管理するが，C 言語ではヒープ領域の確保と解放を明示的にプログラムで指定する必要がある．

3-c. 抽象データ型と情報の隠蔽について、説明しなさい。

答 抽象データ型は，データの仕様 (外部から見える振舞い，インタフェース) を規程することにより，データ型を定義したものであり，具体データ型が，データの実装 (具体的な構成方法) を規程することによりデータ型を定義したものと対照的である．データの仕様と実装を分離し仕様のみを外に見せることにより，外部に影響することなく実装を変更できるという利点がある．情報の隠蔽とは，(様々な意味で使われるが，ここでは) データ型 (あるいはオブジェクト) の実装を外に見せないようにして，実装を変更しやすくするものである．

3-d. オブジェクト指向言語における override (メソッドの上書き) とは、どのような機能であるか、またどのような利点があるか、利用にあたって注意すべき点は何か、など、を説明しなさい。(overload について言及する必要はない。)

答 以下の通りである．クラスの概念をもつオブジェクト指向言語では，上位クラスのメソッド等を，それを継承する下位クラスがそのまま (宣言することなく) 持つが，この際，同一の実装ではなく異なる実装に置きかえることを override(上書き) という．override を許さない場合、継承関係にある下位クラス独自のメソッドは、上位クラスのメソッドと異なる名前を持つ必要があり、プログラムの書きやすさや名前空間の管理が厄介になるが、override を許すことにより、これらの欠点がなくなるという利点がある。静的型システムをもつ Java においては，overrid において，メソッドのインタフェース (引数の個数，引数の型，返り値の型) を変更してはいけない，という制約がある．

また、以下の事項のうち **1 つ以上を選択して**、5 行程度で説明しなさい。(2 つ以上を選択して解答した場合は、特に良い解答の時に加点する。)

4-e. 高階関数、その利点、その処理 (ML など、具体的な言語について論じてもよい)

4-f. 静的型付けと動的型付け (Java と Ruby など、具体的な言語について論じてもよい)

4-g. 多相型 (polymorphic type)

4-h. オブジェクト指向言語における動的ルックアップ (Java など、具体的な言語について論じてもよい)

4-i. オブジェクト指向言語における継承とサブタイピング (Java など、具体的な言語について論じてもよい)

4-j. 汎用言語とドメイン特化言語 (DSL)

4-k. 末尾再帰とは何か、その利点、その処理

これらの問題は、キーワードのみあげて説明せよ、というパターンの問題であるので、講義資料を参考にしてほしい。

以上.