『プログラム言語論』 期末試験 問題と解答例

問 1. (配点 30 点)

次のコードは、あるプログラム言語 (Simple と呼ぶことにする) の構文と意味 (評価器,evaluator) を、OCaml 言語で記述したものである。

なお、これらは、演習で学習したものと、lookup 関数の定義を省略した以外は、同一である。問 1-a 以外の解答は、ここで与えた構文通りでなくてよく、環境の中身がわかる形の記法を使えばよい。(実質的な中身が正しければよい。)

```
type expr =
  | CstI of int
  | Var of string
  | Let of string * expr * expr
  | Prim of string * expr * expr ;;
let rec lookup env x = (* omitted *) ;;
let rec eval e (env : (string * int) list) : int =
  match e with
  | CstI i            -> i
  | Var x             -> lookup env x
  | Let(x, erhs, ebody) ->
    let xval = eval erhs env in
    let env1 = (x, xval) :: env in
       eval ebody env1
  | Prim("+", e1, e2) -> (eval e1 env) + (eval e2 env)
  | Prim("*", e1, e2) -> (eval e1 env) * (eval e2 env)
  | Prim _            -> failwith "unknown" ;;
```

1-a. (5 点) C 言語や OCaml 言語における (2+4) に相当する式を、上記の expr 型の要素として表しなさい。

解答例: 以下の通り： `Prim("+", CstI(2), CstI(4))`

1-b. (5 点) 式 e を、上記の eval で評価したい。eval は e と env を引数として取るが、この env として (eval の最初の呼び出しで) 何を与えればよいか答えなさい。

解答例: env は変数の束縛を表すリストであり、計算の最初の段階ではどの変数も束縛されていないので、空リストを与えるとよい。答え: `[ ]`

1-c. (5 点) 次の expr 型の式を、上記の eval で評価するとき、関数 eval は 再帰呼び出しにより何度か呼ばれるはずである。そこで、eval が呼ばれるごとに引数 e と env がどの値を取るかをすべて (実行の順序に従って) 書きなさい。

`Let("x", CstI(3), Prim("+", Var("x"), CstI(5)))`

解答例: 以下の通り。(この eval の実装では、変数 x が値 v に束縛されていることを、OCaml の対のデータ構造を使って、("x",v) とで表現する。問題文の注釈にある通り、この部分は x=v など、内容がわかる別の表現でもよい。)

1. e=`Let("x", CstI(3), Prim("+", Var("x"), CstI(5)))`, env=`[]`
2. e=`CstI(3)`, env=`[]`

3. e=`Prim("+", Var("x"), CstI(5))`, env=`[("x",3)]`

4. e=`Var("x")`, env=`[("x",3)]`

5. e=`CstI(5)`, env=`[("x",3)]`

1-d. (10 点) 前問と同様のことを、次の式に対して書きなさい。

```
Let("x", CstI(7),
    Prim("+",
         Let("y", CstI(9),
             Prim("*", Var("x"), Var("y"))),
         Let("z", CstI(2),
             Prim("+", Var("z"), CstI(10)))))
```

解答例: 式が長いので、部分式に以下の通り、名前をつける。

e1=`Let("y", CstI(9), Prim("*", Var("x"), Var("y")))`

e2=`Let("z", CstI(2), Prim("+", Var("z"), CstI(10)))`

e と env の値は以下の通り。

1. e=`Let("x",CstI(7),Prim("+",e1,e2))`, env=`[]`

2. e=`CstI(7)`, env=`[]`

3. e=`Prim("+",e1,e2))`, env=`[("x",7)]`

4. e=e1, env=`[("x",7)]`

5. e=`CstI(9)`, env=`[("x",7)]`

6. e=`Prim("*", Var("x"), Var("y"))`, env=`[("y",9);("x",7)]`

7. e=`Var("x")`, env=`[("y",9);("x",7)]`

8. e=`Var("y")`, env=`[("y",9);("x",7)]`

9. e=e2, env=`[("x",7)]`

10. e=`CstI(2)`, env=`[("x",7)]`

11. e=`Prim("+", Var("z"), CstI(10))`, env=`[("z",2);("x",7)]`

12. e=`Var("z")`, env=`[("z",2);("x",7)]`

13. e=`CstI(10)`, env=`[("z",2);("x",7)]`

1-e. (5 点) Simple 言語に print 文を追加した言語では、a+b の形の式 (を Simple の構文に直したもの) の評価で、a から評価するか b から評価するかで、印刷される結果が異なることがある。上記の eval を適宜修正して、「a+b の形の式は a から評価し、a*b の形の式は b から評価する」ようにするためには、どうしたらよいかを述べよ。(OCaml のコードが書ける人は具体的に書けばよい。また、C 言語や Java 言語に似た構文をつかった擬似コードでも構わない。)

解答例: 上記の eval 関数の定義における、加算と乗算の評価は

```
| Prim("+", e1, e2) -> (eval e1 env) + (eval e2 env)
| Prim("*", e1, e2) -> (eval e1 env) * (eval e2 env)
```

の部分である。

したがって、Prim("+", e1, e2) の評価では、(eval e1 env) を評価してから、(eval e2 env) を評価し、そのあと、それらの結果の加算を行ったものを返すようにすればよい。

Prim("*", e1, e2) の評価では、(eval e2 env) を評価してから、(eval e1 env) を評価し、そのあと、それらの結果の乗算を行ったものを返すようにすればよい。

解答例その 2: 具体的な OCaml コードで書くと以下のようになる。

```
  | Prim("+", e1, e2) ->
      let v1 = eval e1 env in
      let v2 = eval e2 env in
         v1 + v2
  | Prim("*", e1, e2) ->
      let v2 = eval e2 env in
      let v1 = eval e1 env in
         v1 * v2
```

[補足] 答案の中には、a+b の処理は、与えられたコードのままとして、a*b の処理だけ b*a の形に変更する、という趣旨のものがいくつかあった。これは、一見正しいように見えて正解ではない。

なぜなら、OCaml 言語では a+b という形の式を a から計算するか b から計算するか未定義 (決まっていない) である。したがって、a+b という形の式を b+a に変更しても、計算順序が逆転するという保証はない。(「未定義」である以上、OCaml 処理系は、ある時は a から計算し、別の時は b から計算してもよいのである。) そのため、確実に a を計算してから b を計算するようにするためには、let x = a in let y = b in x + y といった式にしなければいけない。なお、我々が使っている OCaml 処理系 では、a+b という形の式は、いつでも、b から先に計算される。

本問では、このような OCaml 言語特有の事情に頼らず、言葉で、処理の方針を説明してもらえばよかったのであるが、頑張りすぎて OCaml での実装を書いてしまった人の中には、上記でひっかかった人がいる。(ひっかけるつもりは全くなかったが、a+b という式で a から先に計算されるという前提での答えを正答とするわけにはいかず、若干減点した。)

問 2. (配点 35 点)

今度は、関数をデータとして扱うことのできる言語 (Simple 言語の拡張) を想定して、以下のコードについて考える。

```
Let("x", CstI(3),
 Letfun("f", "y", Prim("+", Var("x"), Var("y")),
   Let("x", CstI(5),
      Letfun("g", "y",
             Prim("+", Prim("*", Var("x"), Var("y")),
                       Var("y")),
        Call(Var("g"), Call(Var("f"),Var("x")))))))
```

参考までに、これは、以下の OCaml コードと同等のコードである。こちらについて考えてもよい。

```
let x = 3 in
let f y = x + y in
let x = 5 in
let g y = x * y + y in
   g (f x)
```

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、プログラムが返す値を示しなさい．また、それぞれの方式での実行において、環境 (前問における env 引数) がどのように変化するか、実行の順序に従って書きなさい。ただし、関数クロージャ等の表記は、自分で設定した表記でかまわない。

2-a. (10 点) 静的束縛かつ値呼び

解答例: 演習で用いた記法通りに書くと長くなるため、環境は `[x=3; y=4]` 等と表し、環境クロージャは、`Closure(y,x+y,[x=3;y=4])` 等と表す。env 引数の変化のみを記載する。

(1) `[]`

(2) `[x=3]`

(3) `[f=Closure(y,x+y,[x=3]); x=3]`

(4) `[x=5; f=Closure(y,x+y,[x=3]); x=3]`

(5) `[g=Closure(y,x*y+y,[x=5; f=Closure(y,x+y,[x=3]); x=3]);x=5; f=Closure(y,x+y,[x=3]); x=3]`

(6) (f の呼び出し) `[y=5; x=3]`

(7) (g の呼び出し) `[y=8; x=5; f=Closure(y,x+y,[x=3]); x=3]`

返す値は 48 である。

2-b. 静的束縛かつ名前呼び

let g y = ... までは、2-a と同じなので、省略する。その時点で、環境は、以下の (5) である。

(5) `[g=Closure(y,x*y+y,[x=5; f=Closure(y,x+y,[x=3]); x=3]);x=5; f=Closure(y,x+y,[x=3]); x=3]`

この環境を、以下では env5 と書くことにする。

次に、g (f x) の処理をするが、名前呼びであるので (f x) を計算せずに y=(f x) という束縛を作って、関数 g の処理にはいる。静的束縛であるので、この (f x) の x は、現時点の x=5 という環境のもとでの x である。一般に、e という式を環境 env のもとで評価するということを、Susp(e, env) と表すことにする。

なお、このあたりの詳細は、授業できちんとは説明していないので、答案では、Susp(f x, [x=5]) を (f 5) 等と書いても差しつかえないものとする。(要するに、名前呼びであることがわかるように書いてあれば良いものとした。)

(6) (g の呼び出し) `[y=Susp(f x, env5); x=5; f=Closure(y,x+y,[x=3]); x=3]`

(7) (f の 1 回目の呼び出し) `[y=Susp(x,env5); x=3]`

f の 1 回目の呼び出しでは 5+3 = 8 が返る。

(8) (f の 2 回目の呼び出し) `[y=Susp(x,env5); x=3]`

2 回目も同様に 8 が返る。

結果として、5*8+8 = 48 が返る。

(ポイントは (6) で y に束縛されるのが、(f x) といった式である点である。(7) や (8) で y に束縛されるものを x でなく、5 とした解答でも正答とした。)

2-c. (10 点) 動的束縛かつ値呼び

動的束縛の場合は、クロージャを作る意味はないので (クロージャに格納する環境を使わないので)、Fun(x,x+y) のように表記することにする。

(1) `[]`

(2) `[x=3]`

(3) `[f=Fun(y,x+y); x=3]`

(4) `[x=5; f=Fun(y,x+y); x=3]`

(5) `[g=Fun(y,x*y+y); x=5; f=Fun(y,x+y); x=3]`

(6) (f の呼び出し) `[y=5; g=Fun(y,x*y+y); x=5; f=Fun(y,x+y); x=3]`

(補足: ここで (5) の環境に y=5 という束縛を追加したことに注意されたい。このため、x=5 という束縛が生きており、f x の計算結果が 10 になる。)

(7) (g の呼び出し) `[y=10; g=Fun(y,x*y+y); x=5; f=Fun(y,x+y); x=3]`

最終的に、5*10+10 = 60 が返る。

2-d. (5 点) 関数呼び出しの方式の 1 つで、必要呼びとは何かを説明しなさい。(その特徴を 2 点以上とりあげ、それぞれ簡潔に説明しなさい。)

解答例: 必要呼びは、関数の実引数を評価せずに、関数に渡す。関数本体の処理において、その引数の値が必要になったときに評価する。そして、1 度評価した引数を再び必要とした時には、再度計算せず、最初の評価結果をそのまま返す。

必要呼びは、名前呼びと類似しているが、引数を 2 回以上使うときは、引数の評価が 1 回だけであるので効率が良くなるという利点がある。

問 3. (配点 25 点)

Java 言語で、2 つのクラス ClassA, ClassB を、以下のように定義する。

```
class ClassA {
  public String toString () { return "ClassA";       }
  public String method1 ()  { return "Method1";      }
  ClassA ()                 {}
}
class ClassB extends ClassA {
  public String toString () { return "ClassB";       }
  public String method2 ()          { return "Method2-1"; }
  public String method2 (String s)  { return "Method2-2"; }
  ClassB ()                 {}
}
```

ClassB は ClassA を継承している。toString, method1, method2 はメソッドである。

このとき、以下のプログラムを実行したとする。ただし、いくつかの行はコンパイルエラーを起こす可能性があり、その場合は、他の行を試すときは、その行をコメントにしているものとする。

また、new ClassA() は、ClassA というクラスのオブジェクトを新たに 1 つ作り、x.method1() は、変数 x に格納されたオブジェクトへのメソッド method1 の呼び出し、System.out.println は引数の値を印刷する。

```
class Test1 {
  public static void main(String args[]) {
  ClassA  x;  ClassB  y;
  y = new ClassA();                 // test1
  System.out.println(y.toString()); // test2
  System.out.println(y.method2());  // test3
  y = new ClassB();                 // test4
  System.out.println(y.toString()); // test5
  System.out.println(y.method1());  // test6
  System.out.println(y.method2());  // test7
  System.out.println(y.method2("a"));  // test8
  x = y;                            // test9
  System.out.println(x.toString()); // test10
  System.out.println(x.method1());  // test11
  System.out.println(x.method2());  // test12
} }
```

3-a. (20 点) 上記の test1 から test12 について、(1) コンパイルエラーか、(2) コンパイルエラーではなく、実行時エラーか、(3) エラーがなく実行すると何かが印刷されるものはその文字列を答えなさい。また、たとえば、test1 がコンパイルエラーの場合、test2 から test3 まではテストできないので、それも (1) に分類せよ。

解答例: 以下の通り。

test1: コンパイルエラー (親クラスのオブジェクトを子クラスの変数に代入できない)

test2-test3: コンパイルエラー (test1 がコンパイルエラーのため)

test4: ok

test5: ”ClassB”

test6: ”Method1”

test7: ”Method2-1”

test8: ”Method2-2”

test9: ok

test10: ”ClassB”

test11: ”Method1”

test12: コンパイルエラー

3-b. (5 点) 上記の例を参考に、Java における override と overload について簡潔に (1-2 行で) 説明しなさい。

解答例: override は、クラスの継承において、親クラスから継承したメソッドの実装を別の実装に置き換えること。この場合、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。

overload は、1 つのクラスの中で、1 つの名前のメソッドを複数定義すること。この場合、メソッドのインタフェースは、異なるものにしなければいけない。

問 4. (配点 15 点)

以下の設問に全て答えなさい。解答の分量の目安は、問題 1 つあたり 5 行程度とする。

4-a. (5 点) コンパイラが最適化 (高速化) のために取得・活用する「プログラムの静的情報」には、どんなものがあるが、具体例をあげて説明せよ。(A という情報は静的であり、これをこういう風に利用すると、コードが高速になる、という形で 2-3 個の例について述べよ。)

解答例: 「プログラムの静的情報」は、実行前 (コンパイル時など) に得られる情報のことである。例としては、(1) 関数 foo が引数 x を持つ場合、その引数が関数内で使われているかどうか、(2) 静的束縛の言語で、変数 x が、環境の中でのインデックス (何番目の変数であるか)、(3) 静的型付けの言語で、式の型 などがある。

コンパイラは、(1) の情報を利用して、使われない引数を削除する (関数に渡さない) コードにすることにより、実行時間が高速化できる。

(2) の情報を利用して、変数の名前を、環境におけるインデックスに置き換えたコードにするこにより、lookup 操作を高速化できる。

(3) の情報を利用して、実行時に、たとえば、加算を行う前に引数が本当に「数」であるかを確認するなどの型チェックをする必要がなくなり、高速化できる。

4-b. (5 点) 静的型付けとは何か、また、静的型付けが、動的型付けに比べて優れている点を (なるべく多く) 述べよ。

解答例: 静的型付けは、プログラムの実行前に (コンパイル時等に) 型の整合性を検査することである。静的型付けと動的型付けはそれぞれ利点、欠点があるが、前者の利点を上げると、(1) 型に起因するエラーが早い段階で (プログラムを実行せずに) 検出できるので、ディバッグの効率があがる。(2) 型に基づいて、モジュールやオブジェクトのインタフェースを記述して、プログラムの抽象化ができる。(3) 型の情報は一種のドキュメントとなり、プログラムが理解しやすく、保守しやすくなる。などがある。

4-c. (5 点) オブジェクト指向言語を特徴付ける 4 つの性質のうち、動的ルックアップ、情報隠蔽、継承とは何かを簡単に説明した上で、なぜそれらがプログラムを書く上で有用な概念であるかを説明せよ。

解答例: 動的ルックアップは、メソッドの名前からメソッド実装を検索する作業を実行時に行うことである。その利点は、既存オブジェクトを拡張して、新しいオブジェクトを実装するというタイプの開発がやりやすくなることがあげられる。

(オブジェクト指向言語における) 情報隠蔽は、クラスのインタフェースと実装を分離して、そのクラスの利用者から、実装を隠すことである。これにより、インタフェースを維持する限り、実装を自由に交換してよいことになり、大規模プログラムの開発を分業しやすくなる。

継承は、既に定義したクラスの実装を、新しく定義した子クラスで引き継ぐことであり、コードの再利用が簡単にできるため、プログラムの開発およびディバッグの効率があがる。

以上.