『プログラム言語論』 期末試験 と解答例

問 1. (配点 40 点+ボーナス加点 5 点) MiniML 言語の具体構文 (あるいは、OCaml 言語) で書かれた次の 2 つのプログラムについて考える。

```
let x = 10 in
let f y = y * y + y in
   f (f x)
```

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

上記の 2 つ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、関数 f が呼ばれる回数を求めなさい。

1-a. 静的束縛かつ値呼び: (解答) 計算結果 12210 と 88, f を呼ぶ回数は 2 と 2

1-b. 静的束縛かつ名前呼び: (解答) 計算結果 12210 と 88, f を呼ぶ回数は 4 と 3

1-c. 静的束縛かつ必要呼び: (解答) 計算結果 12210 と 88, f を呼ぶ回数は 2 と 2

1-d. 動的束縛かつ値呼び: (解答) 計算結果 12210 と 150, f を呼ぶ回数は 2 と 2

1-e. この授業の演習で利用した MiniML 言語の処理系 (インタープリタ) は、上記 1-a (静的束縛かつ値呼び) の実行方式であり、また、環境を受け渡す方式で実装をしている。このことを念頭において、上記の 1 つ目のプログラムを、1-a 方式で実行する過程で、環境がどのように変化するかを書きなさい。(環境が変化するごとに記載しなさい。)

なお、環境の表記は、OCaml の文法に従う必要はなく、[x=10; y=20] や (x=10, y=20) のように、内容がわかる書き方であればよい。

解答: 以下の通り。

[]
[x=10]
[f=Clo(y,y*y+y,[x=10]); x=10]
[y=10; x=10]
[f=Clo(y,y*y+y,[x=10]); x=10]
[y=110; x=10]
[x=10]
[]

ここで Clo は Closure をあらわす。

1-f. 前問と同様に、上記の 2 つ目のプログラムを 1-a 方式で実行する過程で、環境がどのように変化するかを書きなさい。

解答: 概要は以下の通り。

[]
[x=3]
[f=Clo(y,x+y,[x=3]); x=3]
[g=Clo(y,y*(f y),[f=...;x=3]); f=Clo(y,x+y,[x=3]); x=3]
[x=5; g=Clo(y,y*(f y),[f=...;x=3]); f=Clo(y,x+y,[x=3]); x=3]
[y=5; x=3] (関数 f を引数 5 で呼びだした)
[x=5; g=Clo(y,y*(f y),[f=...;x=3]); f=Clo(y,x+y,[x=3]); x=3] ((g 8) の計算)
[y=8; f=Clo(y,x+y,[x=3]); x=3] (関数 g を引数 8 で呼び出した)
[y=8; x=3] (関数 f を引数 8 で呼び出した)
[y=8; f=Clo(y,x+y,[x=3]); x=3] (関数 g に戻った)
[x=5; g=Clo(y,y*(f y),[f=...;x=3]); f=Clo(y,x+y,[x=3]); x=3] (戻った)
[]

ここでは、高階関数の処理において、関数クロージャが必要となることに注意せよ。関数クロージャの表記も、OCaml の文法に従う必要はなく、内容がわかる書き方をすればよい。

1-g. この授業の演習で利用した MiniML 言語の処理系 (インタープリタ) を、上記 1-d (動的束縛かつ値呼び) の実行方式に変更するために、インタープリタを修正したい。どのような修正をすればよいか、言葉で述べよ。(コードを書く必要はなく、インタプリタのどの部分を、どう修正すればよいかを述べなさい。)

解答: MiniML は静的束縛かつ値呼びの言語であり、関数呼び出しにおいては、クロージャの中にある環境 E に、仮引数と実引数の束縛を追加して新しい環境を作っている。

これを、動的束縛かつ値呼びに変更するには、関数呼び出しにおいて、クロージャの中にある環境 E を使わず、現在の環境に仮引数と実引数の束縛を追加して新しい環境を作ればよい。

問 2. (配点 30 点+ボーナス加点 6 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";      }
  ClassB ()                 {}
}
```

ClassB は ClassA を継承した子クラスである。また、それぞれのクラスは toString というメソッドを持ち、それが呼ばれると、クラス名を返す。そのほかに、method1 と method2 というメソッドが定義されている。

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 (型エラーを含む) 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、そのクラスのオブジェクトを新たに 1 つ作る式であり、x.method1() は、変数 x に格納されたオブジェクトに method1 というメソッドを呼びだすものであり、System.out.println は、引数の値を印刷するメソッドである。

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

上記の test1 から test12 について、(1) その行を含めるとコンパイルエラーになるかどうか、(2) コンパイルエラーにならない場合、実行すると実行時エラーになるかどうか、(3) エラーがない場合、実行すると何が印刷されるかを答えなさい。ただし、test1, test5, test9 は印刷文ではないので、(3) を答える必要はない。また、たとえば、test1 がコンパイルエラーの場合、test2 から test4 まではテストできないので、それらも「コンパイルエラー」に分類せよ。

ヒント: Java の型付けは静的であり、メソッド呼び出しは動的ルックアップを用いる。

解答:

test1 ok, test2 ok で ”ClassA”を印刷、test3 ok で ”Method1”を印刷。test4 コンパイルエラー (ClassA に method2 はない)

test5 コンパイルエラー (子クラスの変数に親クラスオブジェクトを代入できない)、test6,test7,test8 コンパイルエラー (test5 が駄目なので)

test9 ok, test10 ok で ”ClassB”を印刷、test11 ok で ”Method1”を印刷。test12 ok で ”Method2”を印刷。

問 3. (配点 30 点+ボーナス加点 6 点)

以下の設問のうち 5 個以上 (1 個を除いて)、答えなさい。解答の分量の目安は、問題 1 つごとに 5 行である (最低 3 行とする)。

4-a. インタープリタとコンパイラの定義を述べよ。また、多くの場合、インタープリタで実行するより、コンパイルしてから実行する方が高速であるが、その理由の主なものはなにか。

略解 (詳細は授業スライドを参照のこと): インタープリタは解釈系であり、プログラムとそれに対する入力データをもらって計算結果を返す。一方、コンパイラは翻訳系であり、プログラムをもらってプログラムを返す。多くのコンパイラは、プログラムの全体あるいはかなり大きな部分 (ファイルやモジュールなどの単位) 全体を読みこんで、その静的情報を集めて解析し、最適化したコードを生成するため、コンパイルされたコードは高速に動作することが多い。一方、多くのインタープリタは、そういった静的情報の収集・解析・最適化を行わないので、コンパイルされたコードほど高速に動作しないことが多い。

4-b. 末尾再帰とは何か述べよ。また、いくつかのプログラム言語では、末尾再帰で記述した方が、非末尾再帰での記述よりも有利である。その理由の主なものはなにか。

略解 (詳細は授業スライドを参照のこと): 関数の再帰呼び出しは、関数 A の定義の中で A 自身を呼びだすことである。また、関数 A の定義の中で B を呼び、関数 B の定義の中で A を呼ぶ場合 (あるいは、3 つ以上の関数がそのような関係にあるとき)、相互再帰と呼び、再帰呼び出しの一種と考える。このような再帰呼び出しの中でも、末尾再帰とは、関数の再帰的な呼び出しが全て、「末尾」の位置であるものである。末尾の位置とは、式の計算において、「そのあとに計算が残っていない」位置のことであり、たとえば foo(goo(hoo(x))) の計算において、関数 goo や hoo の呼び出しは末尾でなく、foo の呼び出しは末尾である。末尾再帰は、関数の処理後に呼び出し元にもどる必要がないので、実装上は、一方向のジャンプ命令で実現することができ、また、関数の局所変数などの環境 (スタックに積まれるもの) を保存する必要がなくなり、速度および使用メモリ量の両面で、非末尾再帰より有利である。

関数型言語は、for, while などのループではなく、再帰呼び出しを多様するプログラミングスタイルを取ることが多く、このため、末尾再帰の最適化を行う処理系が多い。たとえば、Scheme, OCaml などはこの最適化を行う。

4-c. 静的型付けと動的型付けの定義と違い (特にそれぞれの利点と欠点)、また、型検査と型推論の定義と違いを述べよ。

略解 (詳細は授業スライドを参照のこと): 静的型付けは、プログラムに対する型の整合性の検査 (また、場合によっては、型の推論) を静的に、つまり、実行前に行うことである。一方、動的型付けは、実行前ではな、実行時にプログラムに対する型の整合性の検査を行う。

静的型付けは、プログラムのバグを早期に発見することができ、型付けに成功したプログラムは、どのような実行においても型エラーを起こさないことが保証されるため、安全性、信頼性に優れる。動的型付けは、その逆であるが、プログラム全体にわたって型の制約を守る必要が必ずしもないため、プログラム開発過程やプロトタイププログラムの作成においては、動的型付けの言語の方が優れているという主張もある。

型検査は、プログラムと型、型文脈が与えられたとき、それらが整合するかを判定することである。C, Java などのプログラミング言語は型検査をコンパイル時におこなう。型推論は、プログラムが与えられたとき、それが型付け可能となるような型文脈と型を推論することである。OCaml, Haskell などのプログラミング言語は型検査をコンパイル時におこなう。

4-d. 多相型とは何か述べよ。(ここでは、一番基本的な多相、つまり、パラメータ多相について答えればよい。オブジェクト指向言語におけるサブタイプ多相や、アドホック多相について答える必要はない。) 多相型を用いることによるプログラミング上の利点と、利用例を 1 つ述べよ。

ヒント: 「利用例」というのは、たとえば、ソートアルゴリズムで多相型をどのように使うと良いか、利用シーンを 1 つ想定して書くとよい。

略解 (詳細は授業スライドを参照のこと): 多相型は、1 つの関数 (あるいはデータ) を複数の型の要素に適用可能とするものである。多相型のなかでも、パラメータ多相は、1 つの関数が、型パラメータをもつ型を持つことであり、この型パラメータ (型変数) は、「そこにどんな型を代入してもよい」ということをあらわす。すなわち、任意の型にたいして、あるパターンの型をもつ関数のことをパラメータ多相型をもつ関数という。

多相型は、1 つの関数が型ごとに定義されなくても済むため、プログラムの抽象化や再利用性等に役立つ概念である。プログラミングにおいては、多相型関数を上手に使うことにより、プログラムのコード量を減らしたり、見通しのよいプログラムを構成することができる。

多相型の利用例のひとつは、リストに対する操作である。リストに対する結合などの演算の多くは、パラメータ多相型を持ち、リストの要素型がなんであっても同じコードで対応できる。

4-e. 情報の隠蔽 (情報のカプセル化、抽象化) とは何か述べよ。また、そのような考えがなぜ有用であるのか、現在どのように利用されているか、などを論じよ。

略解 (詳細は授業スライドを参照のこと): データ抽象化やオブジェクト指向言語における「情報の隠蔽」とは、データやオブジェクトのインタフェースと実装を明確に分離し、外部に見せるものはインタフェース部分だけとすることにより、外部に影響を与えることなく実装を変更することが可能になり、大規模ソフトウェアを分業により作成するときに、特に大きな威力を発揮する。

4-f. 静的な情報と動的な情報の違いの概要を答えなさい。また、次のうち、「静的に決定できる情報」がどれか答えなさい。

(1) Java 言語あるいは C 言語において、プログラムで宣言されている変数がすべて、プログラムの中で一回以上出現するか？(2) Java 言語あるいは C 言語において、プログラムで宣言されている変数がすべて、プログラムの中で実際に一回以上使われているか？(3) OCaml 言語 (あるいは MiniML 言語) で、プログラムに出現する変数がすべて、int 型を持つか？

解: (1) と (3)。

(1) はプログラムのテキストを見ただけで決定できる情報であるので静的である。

(2) は静的には決定できない。たとえば、if L=L then x+1 else 10 という式において、変数 x が使われるかどうかは L=L が true であるか false であるかに依存して決まる。この場合、L の計算が止まるなら L=L は成立するので x の値は使われ、L の計算が止まらないなら (そこで無限ループなので)、x の値は使われない

しかし、L は一般の式であるので、それが停止するかどうかを静的に決定することはできない。

(3) は、OCaml においては、変数の型を静的に推論することができ、推論した結果の型が int になるかかどうかをチェックすればよいので、上記 (3) は、全体として、静的に決定できる。

以上.