Programming Languages 3: Questions and Answers: April/May 2011
Duration: 90 minutes.
Rubric: Answer all four questions. Total 60 marks.

1. (*Functional programming*)

(a) Given a list of integers, `ns`, write the following:

(i) a list comprehension that yields a list of Boolean values, in which each element is `True` if the corresponding element of `ns` is even, or `False` if the element is odd;

(ii) a list comprehension that yields a list of integers, in which each element is one less than the corresponding element of `ns`, except that non-positive elements of `ns` are discarded.

[Similar to seen problem]

(i)
```
[mod n 2 == 0 | n <- ns]
```

(ii)
```
[n - 1 | n <- ns; n > 0]
```

[1+1]

(b) Given a list of Boolean values, `bs`, define the following functions:

(i) "`all bs`" yields `True` if and only if `bs` is empty or all its elements are `True`;

(ii) "`any bs`" yields `True` if and only if `bs` is non-empty and at least one of its elements is `True`.

Make sure that your functions are efficient.

Your answer must explicitly declare the type of each function. *Do not use any list functions from the Haskell standard prelude*.

[Unseen problem]

(i)
```
all :: [Bool] -> Bool
all []     = True
all (b:bs) = b && all bs
```

(ii)
```
any :: [Bool] -> Bool
any []     = False
any (b:bs) = b || any bs
```

[Alternative solutions using if…then…else… are equally acceptable. But lose 1 mark each if short-circuit evaluation is not used.]

[2+2]

(c) Given an arbitrary list, `xs`, define the following polymorphic functions:

(i) "`map f xs`" yields a list in which each element is the result of applying function `f` to the corresponding elements of `xs`;

(ii) "`filter f xs`" yields a list containing just those elements of `xs` satisfied by function `f` (i.e., those elements `x` for which `f x` yields `True`).

Your answer must explicitly declare the type of each function. *Do not use any list functions from the Haskell standard prelude*.

[Unseen problem]

(i)
```
map :: (a -> b) -> [a] -> [b]
map f []     = []
map f (x:xs) = f x : map f xs
```

(ii)
```
filter :: (a -> Bool) -> [a] -> [a]
filter f []     = []
filter f (x:xs) =
    if f x
    then x : filter f xs
    else filter f xs
```

[Answers using list comprehensions are equally acceptable.]

[4+5]
[*total 15*]

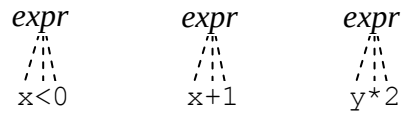
2. (*Syntax*)

Box 1 shows part of the BNF grammar of a fictional programming language, FPL. It shows only the syntax of commands and sequential commands. (The syntax of expressions is not needed in this question, and is not shown in Box 1.)

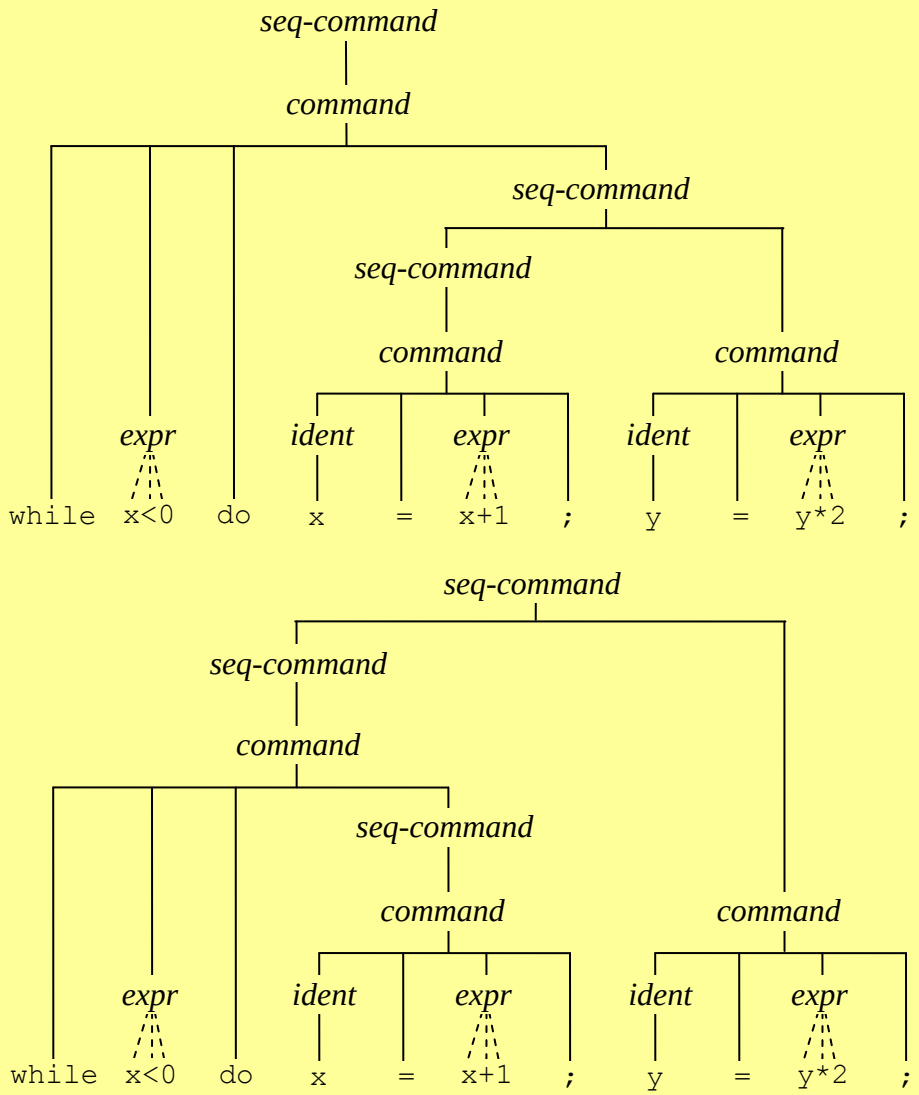
(a) By drawing two different syntax trees, show that the following FPL sequential command is ambiguous:

```
while x<0 do x = x+1; y = y*2;
```

You may assume that `x<0`, `x+1`, and `y*2` are expressions. Your syntax tree should show these expressions in outline:



[Similar to seen problem]



[6]

(b) Suggest how the grammar might be modified to eliminate the ambiguity. Your modified grammar must still allow a sequential command within a loop body.

[Unseen problem]

command ::= *ident* = *expr* ;
| `while` *expr* `do` *seq-command* `end`
| …

or:

command ::= *ident* = *expr* ;
| `while` *expr* `do` *command*
| `begin` *seq-command* `end`
| …

[4]

[*total 10*]

```
seq-command ::= command
              | seq-command command

    command ::= ident = expr ;
              | while expr do seq-command
              | ...
```

Box 1 Part of the grammar of a fictional programming language FPL. (Here *expr* is an expression, and *ident* is an identifier.)

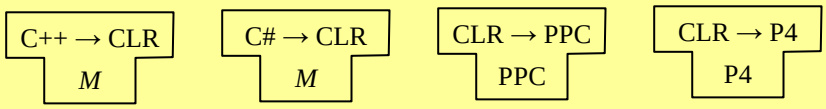
3. (*Implementation*)

(a) The Microsoft common-language run-time system (known as CLR or .NET) consists of a suite of front-end compilers translating high-level languages into CLR intermediate code, plus a suite of back-end compilers translating CLR code into native machine code. The front-end compilers run on the software developer's machine, but the back-end compilers run on the customer's machine.

Using this system, a software developer writes an application program in a high-level language, compiles it, and distributes it as CLR code.

Draw tombstone diagrams for (i) a C++ front-end compiler; (ii) a C# front-end compiler; (iii) a PPC back-end compiler; (iv) a P4 back-end compiler.

[Seen problem]



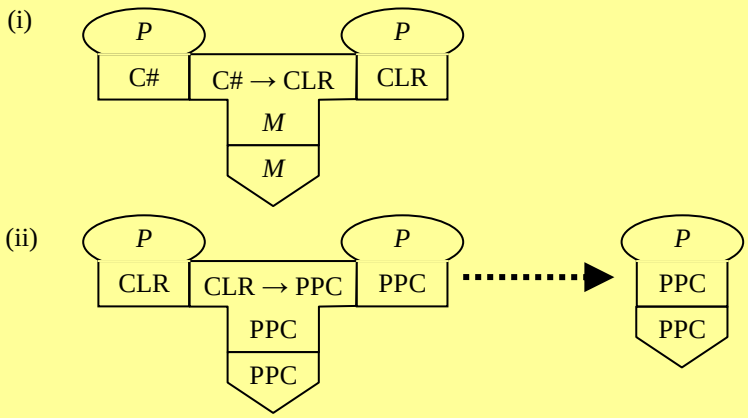
where *M* is the developer's machine.

[4]

(b) Using tombstone diagrams, show how an application program *P* written in C# would be (i) compiled, and (ii) run on a customer's PPC machine.

Comment on your answer to (ii) – when exactly would PPC native code be generated?

[Similar to seen problem]



In (ii), PPC code could be generated *either* just-in-time (each time *P* is run) *or* once and for all (when *P* is installed or when *P* is first loaded). [Either answer is acceptable.]

[2+2+1]

(c) Compare and contrast the CLR system with the Java Development Kit. What are the strengths and weaknesses of each?

[Insight]

CLR supports multiple languages and multiple platforms. JDK supports multiple platforms but only one language, Java.

CLR supports only compilation. JDK supports a choice of interpretation or JIT compilation, which is good because interpretation is preferable during program development and for infrequently-executed code.

[6]

[*total 15*]

4. (*Concepts*)

(a) What is meant by an *abstract data type* (*ADT*)?

[Notes]

An ADT is a type characterized by its values and operations only. Its data representation is hidden.

[2]

(b) Explain briefly how ADTs are supported by (i) Java and (ii) Haskell.

Illustrate your answer by outlining (i) a Java program-unit and (ii) a Haskell program-unit, each implementing an ADT whose values are sets of integers. It must be possible to construct an empty set, to add a given integer to a set, and to test whether a given integer is in a set. The data structure representing the set must be hidden.

Your outlines should show the headings of all operations, but should not show the data structure or how the operations are implemented.

[Unseen problem]

(i) Java: Use a class whose instance variables are specified as private but whose constructors and methods are specified as public.

```
class IntSet {
    // An IntSet object represents a set of integers.

    private …  // data structure

    public IntSet () { … }
    // constructs an empty set

    public void add (int x) { … }
    // adds x to this set

    public boolean contains (int x) { … }
    // returns true iff x is in this set

}
```

(ii) Haskell: Use a module whose heading lists the type itself and its public operations. Declare the type using a data declaration, but keep its constructor(s) private.

```
module IntSets (IntSet, empty, add, contains)
where

    data IntSet = SET …  -- data structure
    -- An IntSet value represents a set of integers.
```

```
empty :: IntSet
empty = …
-- the empty set

add :: Int -> IntSet -> IntSet
add x s = …
-- adds x to set s

contains :: Int -> IntSet -> Bool
contains x s = …
-- yields true iff x is in set s
```

[4+4]

(c) Show how you would generalize your Java and Haskell program-units of part (b) to supports sets of elements of an arbitrary type.

[Unseen problem]

(i) Java: Make the class generic.

```
class Set <T> {
    // A Set<T> object represents a set of objects of type T.

    private …  // data structure

    public Set<T> () { … }
    // constructs an empty set

    public void add (T x) { … }
    // adds x to this set

    public boolean contains (T x) { … }
    // returns true iff x is in this set

}
```

(ii) Haskell: Make the module export a polymorphic type and polymorphic operations.

```
module Sets (Set, empty, add, contains)
where

    data Set t = SET …  -- data structure
    -- A Set t object represents a set of values of type t.

    empty :: Set t
    empty = …
    -- the empty set

    add :: t -> Set t -> Set t
    add x s = …
    -- adds x to set s
```

```
contains :: t -> Set t -> Bool
contains x s = …
-- yields true iff x is in set s
```

[3+3]

(d) Suppose that an additional requirement is now introduced: it must be possible to obtain the *least* value in a set. Show how your Java program-unit of part (c) would be modified.

[Unseen problem]

Insist that `T` implements the `Comparable<T>` interface. Add a `least()` method that uses `compareTo()` to compare values of type `T`.

```
class Set <T extends Comparable<T>> {
    // A Set<T> object represents a set of objects of type T.

    private …  // data structure

    public Set<T> () { … }
    // constructs an empty set

    public void add (T x) { … }
    // adds x to this set

    public boolean contains (T x) { … }
    // returns true iff x is in this set

    public T least () { … }
    // returns the least element of this set

}
```

[4]

[*total 20*]