SCHOOL OF MATHEMATICAL AND COMPUTER SCIENCES

Computer Science

F28PL

Programming Languages (mock exam)

Semester 1 201516

Duration: Two Hours

This mock exam is aimed slightly harder than the real exam, because doing real exams is harder in real exam conditions because of the stress.

If you ace this mock then you should pass the exam without difficulty. If you don't understand something in this mock, then make sure you understand it . . . soon!

This mock is available in two versions: one with answers and one without. For best results, try to do the one without answers first (you'll learn more from reading the answers if you've made a serious effort to figure it out yourself).

1. (a) Clearly write the ML types of the following expressions, or if the expression has no ML type, explain why: (5)

1. `[0,1,2,3]`
2. `[0.0,1.0,1]`
3. `0 div 0`
4. `fn f => f f`
5. `fn (f,g,x) => g(f(x))`

(b) State the type of the following ML program, and explain what function is calculated by it, making specific reference to the ML execution model (in other words: convince the examiner you understand not only what the program computes, but how):

```
exception Break;
fn f => fn a => fn b =>
   (f(raise Break)
      handle Break => if (f a) then a else raise Break)
         handle Break => if not(f a) then a else b;
```

(3)

(c) Write ML functions of the following types:

1. `('a -> 'b) -> ('a -> 'c) -> 'a -> ('b*'c)` (2)
2. `('a * 'b) -> ('a -> 'c) -> ('b -> 'd) -> 'c*'d` (2)
3. `'a -> 'b list` (2)

(d) The **logistic map** is specified by

$$x_0 = 0.5 \quad \text{and} \quad x_{n+1} = r x_n (1 - x_n)$$

where $x_0, x_1, x_2, \ldots$ is a sequence of reals and r is a real number. The logistic map is (part of) the basis of *chaos theory*.

1. Write an ML program

   `logistic : real -> int -> real`

   that if given arguments `r:real` and `n:int` will compute x_n (for the given value of `r`). Answers that do not respect ML's strict type system may lose marks. (3)
2. Write an ML program

   `list_f : (int -> 'a) -> (int -> 'a list)`

   that if given `f:int->'a` and `n:int` computes $[f(0), f(1), \ldots, f(n)]$. (2)

3. Using your answers above, write a program

   `logistic_map : real -> int -> real list`

   that if given `r:real` and `n:int` computes $[x_0, x_1, \ldots, x_n]$. (1)

2. (a) State the output of the following programs and explain why, or, if the program terminates with an error state what that error is and why it arises:

1. `"Hello dolly"[::-1]` (2)
2.
```
"".join([x[0] for x in
   "Young Men's Christian Association".split(" ")])
```
(2)
3. `["Hello"].append(["world!"])` (2)
4.
```
x=[]
for i in range(4):
   x=[x]*(len(x)+1)

x[3][2][1][0]
```
(2)

(b) Consider the following Python3 code:

```
1 fun mystery d:
2    L = []
3    for k in d:
4       if d[k] not in L:
5          L.append(d[k])
6 return(sorted(L))
```

1. The program is defective and contains four errors. State what they are and how to correct them. (4)
2. Describe what function the program calculates. Clearly state any typing assumptions that you make of the input `d`.
   Note that we do not want a blow-by-blow account of execution: we want to know mathematically what it calculates, or to put another way, how you might document the program for a user. (2)
3. The function `mystery` can be expressed in one line of code, using lambda. Propose how. (2)

(c) Consider the following Python code:

```
x=["Live, Die, "]
x.extend([x])
while True:
   print(x[0],end="")
   x=x[1]
```

Describe the data structure stored in `x` when execution is at line 3. (2)

(d) Describe and explain the output of the program. (2)

3. (a) Compare and constrast the following terms in detail. Where appropriate illustrate your explanation with concrete code samples, being clear about which language you intend to be writing in. Prove to the examiner that you not only understand these terms, but understand their concrete relevance to specific code of the languages in this course. Note the number of marks for the questions: these give some indication of a minimum of how many individual points you should make in each answer.

1. Functional, logic, and imperative programming. (3)
2. Global and local state. (2)
3. Mutable vs. immutable variable. (2)
4. Mutable vs. immutable type (in Python). Be specific giving at least one example of each. (4)
5. Ad hoc polymorphism, and parametric polymorphism. Be specific and give at least one example of each. (4)
6. Dynamic type error and static type error. (2)

(b) Imperative programming is doomed: in ten years we'll all be using pure functional programming. Discuss, giving at least two points for and two points against. (4)

(I know this question has more than 20 points. With so many interesting questions to set, I couldn't decide what to cut.)

4. (a) Explain the differences in Prolog between the *static* and the *dynamic* databases. Your answer should make clear the usage and meaning of the `assert` and `retract` keywords, and should be specific about where and when they can be used. (3)

(b) Convert the following list of English sentences into a Prolog database:

1. If I have chocolate, then I want chocolate.
2. If I want chocolate, then I buy chocolate.
3. If I want chocolate and I have chocolate, then I eat chocolate.

(3)

(c) The French word *sera* means 'will be', and the French word *que* means 'whatever'. Thus the French saying *que sera, sera* can be translated into English as *what will be, will be*, and into Prolog as

```
sera(X) :- sera(X).
```

Explain the behaviour of Prolog when asked to predict whether there will be world peace by asking the query `sera(world_peace)` in this database. Your answer should demonstrate specific understanding of the Prolog execution model. (2)

(d) Now consider the following database:

```
sera(Y) :- sera(X).
```

Explain the behaviour of Prolog when asked to predict whether there will be world peace by asking the query `sera(world_peace)` in this database. Your answer should demonstrate specific understanding of the Prolog execution model. (3)

(e) Write a Prolog program `sumsq` to calculate the sum of squares of a list of integers (that is, the sum of the list divided by its length). Thus `sumsq([3,4],25)` should return `true` and `sumsq([0,-1,1],X)` should return `X=2`. (4)

(f) Consider the following two databases:

- Database 1:

```
food(chicken).
food(fish).
eat(X) :- food(X),!.
```

- Database 2:

```
food(chicken).
food(fish).
eat(X) :- !,food(X).
```

Describe and explain the behaviour of the query `eat(X)` in each database, with specific reference to the Prolog execution model. (2)

(g) Note that `writef(string)` prints `string` to standard output and then succeeds. With an empty database we type the following at the interactive prompt:

```
assert((eat(chocolate)  :- want(chocolate), have(chocolate),
                           writef("Chocolate face!"),
                           retract(have(chocolate)))).
assert((want(chocolate) :- buy(chocolate))).
assert((buy(chocolate)  :- assert(have(chocolate)))).
eat(chocolate).
```

What will Prolog do, and why? (2)

(h) Suggest a simple modification to the code above that will cause it to print `Chocolate face!` forever. (1)

EAT CHOCOLATE