UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Exam in:	**INF3110/4110 Programming Languages**
Day of exam:	**4. December 2007**
Exam hours:	**14:30 – 17:30**

This examination paper consists of 10 pages including page 9 that is used for answer to question 1a, and page 10 that may be used for sketching this answer.

Appendices:	**No**
Permitted materials:	**All printed and written**

Make sure that your copy of this examination paper is complete before answering.

This exam consists of 3 questions that may be answered independently. If you think the text of the questions is unclear, make your own assumptions/interpretations, but be sure to write these down as part of the answer.

Good luck!

Contents

Question 1 Runtime-systems, scoping, types (weight 40%)

1a

NB.: This part of question 1 is answered by filling in values and links in the figure at page 9, and deliver it together with the rest of the answers! Remember to fill in candidate number and date!

Some smart guy wants to add functions as parameters to Java. He assumes that doing it for method parameters will be easy, so he starts out by trying to introduce functions as parameters to classes. He starts out with the following rules:

1. only functions with no return type and with no parameters are allowed;
2. a function as parameter becomes a method of the class with the parameter;
3. actual parameters are provided as part of the generation of objects;
4. actual parameters can be methods that are visible in the block containing the generation of the object.

Below is a simple example according to these rules. The class `C` has a single formal function parameter `f`, this is in method `mc` called as if it was a local method of class `C` and in the call `rc.f()` it is called via a reference typed by `C`. The default constructor for a class with function parameters will take a list of functions as actual parameter and bind the formal function parameters to these actual functions – one does thus not need to specify such a constructor.

```
class Program {
  public static void main(String[] args) {
    D rd = new D(); rd.md();
  }
}
class C (void f()){
  int i = 0;
  public void mc(){
    i = i + 1;
    f();
  }
}
class D {
  int i = 0;
  C rc;
  public void g(){
    i = i + 2;         // *
  }
  public D(){          // constructor for D
    rc = new C(g);
  }

  public void md(){
    rc.f();
    rc.mc();
  }
}
```

The program execution is started by execution of the `main` method in the class `Program`. We assume that the language has static scoping.

Fill in access and control links and values of variables of activation records and objects at the stage of execution when the activation record for `g` is on top, the activation record for `mc` is the second to the top of the run time stack, and the statement marked with * has been executed. Fill in also the links of the only closure that is needed, and indicate how the parameter `f` is represented. Note that access links may not only be links to activation records.

1b

Having done this successfully, the language is extended with parameters of functions as parameters, and the following rule is introduced:

- A method is an allowed actual parameter if the types of the parameters of the method are the *same* types or *subtypes* of the corresponding parameter types of the formal function.

The following example is made according to this rule: `g(ColorPoint cp)` is a valid actual parameter, because the type of the parameter `cp` is a subtype of the parameter type `Point` for `f`. Java had to be extended such that the variables in the `main` method (here `rp`, `rCP` and `rd`) are visible in all other classes. Details like constructors and the definition of the class `Point` are not included.

```
class Program {
  public static void main(String[] args) {
    Point rP = new Point(1,2);
    ColorPoint rCP = new ColorPoint(1,2,red);
    D rd = new D(); rd.md();
  }
}
class Point {int x,y; ...}
class ColorPoint {
  Color c;
  Color getColor {return c;}
}

class C (void f(Point)){
  int i = 0;
  public void mc(){
    i = i + 1;
    f(rP);
  }
}
class D {
  int i;
  C rc;
  public void g(ColorPoint cp){
    Color c = cp.getColor();
    i = i + 2;
  }
  public D(){
    rc = new C(g);
  }

  public void md(){
    rc.f(rCP);
    rc.mc();
  }
}
```

a)

It is not possible to statically type check this, and the program has a runtime type error. Explain why and tell which type error occurs.

b)

Would it be possible to solve this by explicit casting? Answer Yes or No, with a few lines of explanation.

c)

Would another value of `rP` make the program not have a runtime type error? If Yes, please indicate the value, if No, explain why.

Question 2 ML (weight 40%)

2a Type inference

Consider the function:

```
fun f1(x::xs , y::ys) = (x , y) :: f1(xs,ys)
  | f1([],[]) = [] ;
```

a)

Explain what the function f1 does.

b)

What is the type of f1? Explain informally and in a few sentences why f1 has this type.

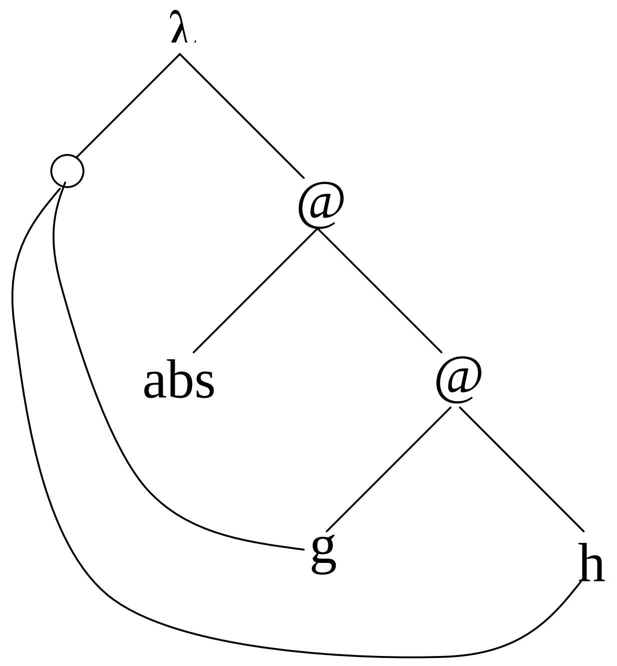
c)

Infer the type of the following function:

```
fun f2(g,h) = abs(g(h))
```

where abs is the function that calculates the absolute value of an int, and is of type: `int -> int` .

Complete the parse tree below by following the steps of the ML type inference algorithm, explicitly describing the 3 steps of the algorithm.



2b Programming with lists

Note that your ML-function definitions must be purely functional, imperative constructs are not allowed, specifically you may not use reference types or while loops. In the definition of functions you may use functions which have been mentioned earlier in the exam, or which you have been asked to define earlier in the exam even if you did not manage to define them. You may not use predefined or library functions unless it is explicitly mentioned that you may do so.

We first consider functions that take a list of integer pairs as input, as for example:

```
val mypairs = [(0,0),(1,0),(1,3),(3,3)] : (int * int) list
```

a)

Define the functions

```
fun getEquals(pairs: (int * int) list):(int * int) list = ...
fun sumPairs(pairs: (int * int) list): int list = ...
```

getEquals should return a list containing only those pairs where the first and second element in the pair are equal and the function sumPairs should return a list with the sums of each pair in the input list.

Example:

```
getEquals(mypairs)    evaluates to    [(0,0),(3,3)]
sumPairs(mypairs)     evaluates to    [0,1,4,6]
```

b)

The higher order functions map and filter have the following definitions in ML:

```
fun map f nil = nil
  | map f (x::xs) = (f x) :: map f xs ;

fun filter p nil = nil
  | filter p (x::xs) = if p x then x :: (filter p xs) else filter p xs ;
```

Redefine the function getEquals by using filter and redefine the function sumPairs by using map. (Hint: remember that the op keyword can be used to turn an infix operator into a function, for example instead of writing 1 < 5 you may write (op <)(1,5) .)

c)

The List constructors in ML are nil (or []) and the infix operator cons (::). When consing a value to a list it is added to the start of the list i.e.:

```
1::[3,4,5] = [1,3,4,5]
```

Define the ML-function

```
snoc : 'a * 'a list -> 'a list
```

that adds an element to the end of a list instead, i.e.:

```
snoc(1,[3,4,5])    evaluates to    [3,4,5,1]
```

2c Records

Recall that in the mandatory assignment, we could use a record type to keep track of the program state (the values of the variables in the program during execution). Here we consider a very simple state with only two variables:

```
type state = { i : int , j : int };
```

The execution of a program leads to a list of states as e.g.:

```
val mystates=[{i=0,j=0},{i=1,j=0},{i=1,j=3}];
```

Instead of a list of states we would like to have a *record* containing two lists which show how the variables change during execution, so we define the function:

```
fun listToRecord(states) = listToRec(states,{il=nil, jl=nil});
```

The function listToRecord takes a list of states as input and calls the function listToRec with this list and a record containing two empty lists. Evaluating

```
listToRecord([{i=0,j=0},{i=1,j=0},{i=1,j=3}])
```

should give the result:

```
val res = {il=[0,1,1], jl=[0,0,3]} : {il:int list, jl:int list}
```

Define the ML-function

```
listToRec : state list * {il:int list, jl:int list} -> {il:int list, jl:int
list}
```

such that the call to listToRecord gives the correct result.

Question 3 Prolog (weight 20%)

Here are some facts about some of the descendants of King Olav V of Norway. The predicate `regent(X)` declares that we know that `X` is/was king. The predicate `royal(Name, Gender, Parent, Born)` declares that somebody with the given name and gender (`Gender` is either `male` or `female`) was born in the year `Born`. `Parent` is the one of the person's parents who is a descendant of Olav V. (One of the parents must be if the person is a descendant of Olav V, but only one can be, since there is no incest in this family.)

```
regent(olav).

royal(ragnhild, female, olav, 1930).
royal(astrid, female, olav, 1932).
royal(harald, male, olav, 1937).

royal(maertha-louise, female, harald, 1971).
royal(haakon, male, harald, 1973).

royal(ingrid-alexandra, female, haakon, 2004).
royal(sverre-magnus, male, haakon, 2005).
```

```
  royal(maud-angelica, female, maertha-louise, 2003).
  royal(leah-isadora, female, maertha-louise, 2005).
```

3a

Write a query to list all known male members of the royal family.

3b

Write a rule/rules to define the following predicates:

- `male(X)`, that is true if `X` is male
- `female(X)`, that is true if `X` is female
- `child(X,Y)`, that is true if `X` is a child of `Y`.
- `descendant(X,Y)`, which is the reflexive, transitive closure of child, i.e., it holds if `X` is the same as `Y`, or `X` is the child of some `Z` who is a descendant of `Y`.
- `older(X,Y)`, that is true if `X` is older than `Y`. (We assume that comparing the years is sufficient).

3c

Write a rule/rules defining a predicate `candidate(X)` that decides whether `X` is a candidate for succession to the throne. Traditionally, all and only the male descendants of some regent were candidates, but in 1990 the succession law of Norway was modified to allow also women born in or after 1971, i.e. Märtha Louise is a candidate, but Ragnhild isn't. Your definition should take this into account.

3d

In the family tree given by the "`royal`" clauses, all family members share at least one common ancestor, namely Olav V. In fact, any two family members have a uniquely defined youngest common ancestor. Define a predicate `yca(X,Y,A)` that is true if `A` is the youngest common ancestor of `X` and `Y`. You can use the following observation:

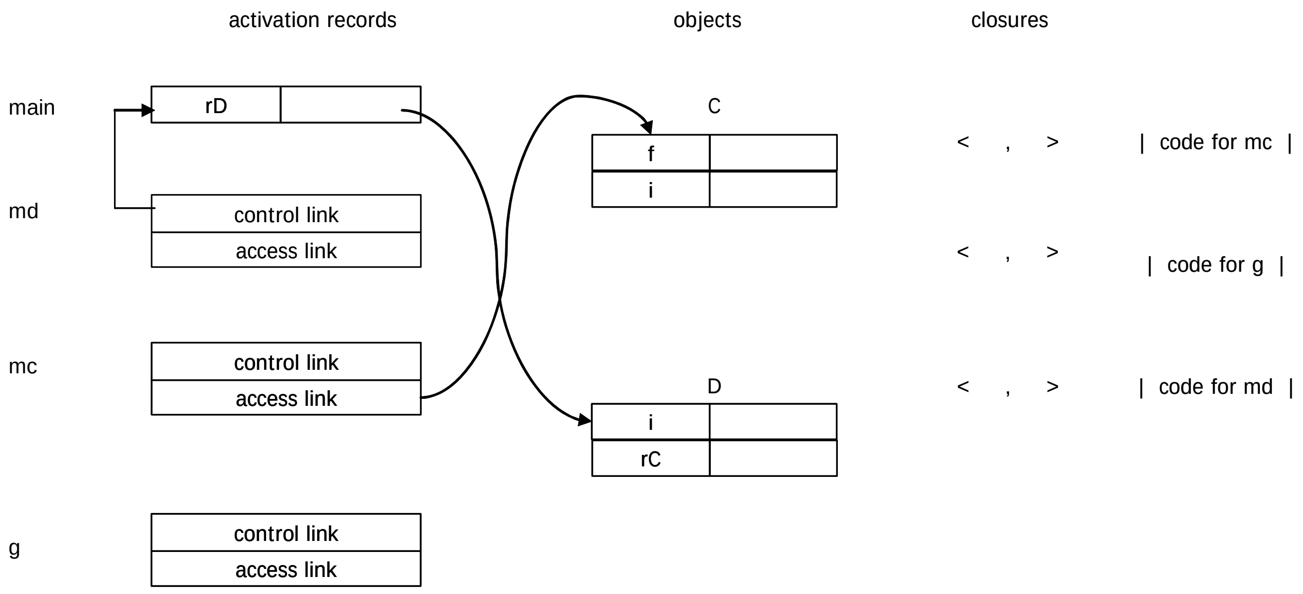
The age of every person is known, and it is safe to assume that each person is younger than all of his predecessors. Therefore, to find the youngest common ancestor of `X` and `Y`,

- If `X=Y`, then the youngest common ancestor is `X` itself.
- Otherwise, take the parent `P` of the younger of `X` and `Y`, and look for the youngest common ancestor of `P` and the older of `X` and `Y`.

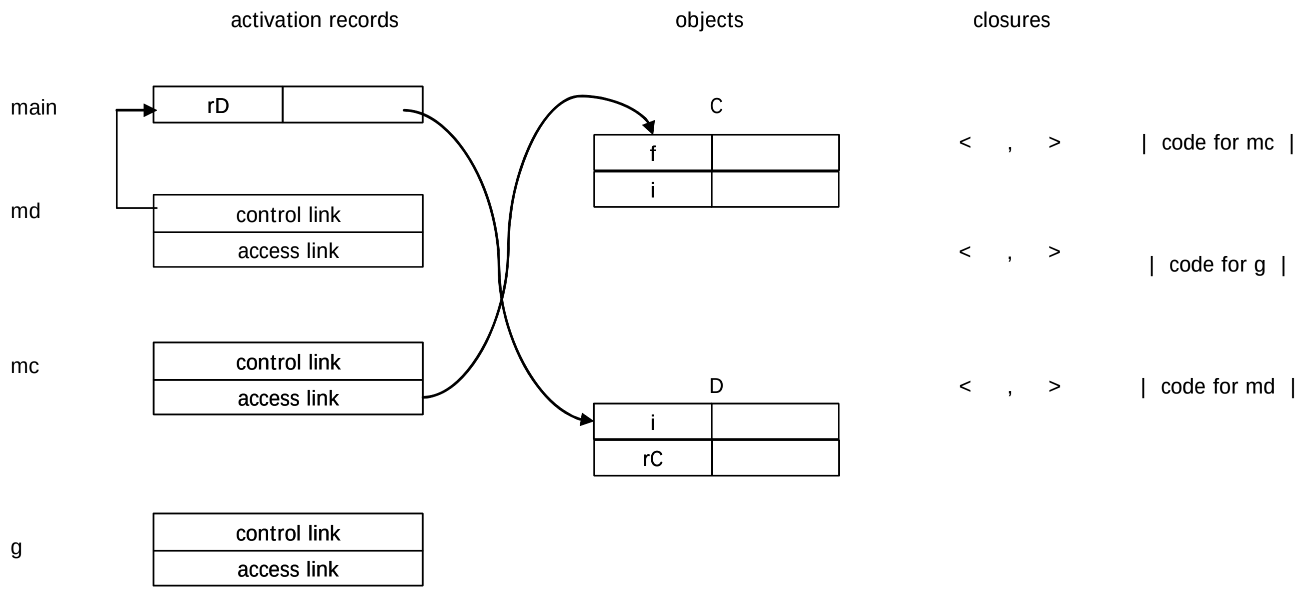
Page for answering Question 1a

Candidate no: ....

Date: ....



Extra page for sketching the answer to Question 1a


activation records
objects
closures
main
rD
md
control link
access link
mc
control link
access link
g
control link
access link
C
f
i
D
i
rC
< , >
| code for mc |
< , >
| code for g |
< , >
| code for md |