

**Possible solutions for
Written examination
19 June 2002**

Written 2004-06-03

The suggested solutions below are not the only possible ones.

Question 1 (30 %): Standard ML

Question 1.1

```
> val sum = fn : int list -> int
    val sumlist = fn : int list list -> int list
    val sumsumlist = fn : int list list -> int
    val ('a, 'b) map = fn : ('a -> 'b) -> 'a list -> 'b list
    val ('a, 'b) maplist = fn : ('a -> 'b) -> 'a list list -> 'b list list
```

The application `sum xs` returns the sum of the elements of `xs`. (The overloading resolution defaults to `int`, but with a suitable type constraint the function could work on `real` instead; this is not an important point in the question). The application `sumlist xss` returns a list of the sums of the individual integer lists in `xss`. The application `sumsumlist xs` returns the sum of the integers in all the integer lists in `xss`. This is the same as `sum (sumlist xss)`. The application `map f xs` returns the list of results of applying function `f` to each element of `xs` from left to right. The application `maplist g xss` returns the list of lists of results of applying function `g` to each element of the individual lists in `xss`, from left to right.

Question 1.2

```
fun doublelist xss = map (map (fn x => 2*x)) xss
```

Question 1.3

```
fun sorted [] = true
  | sorted [t1] = true
  | sorted (t1 :: t2 :: rest) = t1 < t2 andalso sorted (t2::rest);

fun sortedlists xss = List.all sorted xss
```

Question 1.4

```
fun merge [] ys = ys : int list
  | merge xs [] = xs
  | merge (x::xr) (y::yr) =
    if x<y then x :: merge xr (y::yr)
    else if y<x then y :: merge (x::xr) yr
    else x :: y :: merge xr yr;
```

Question 1.5

```
fun mergelists [] = []
  | mergelists (xs::xsr) = merge xs (mergelists xsr);
```

Question 1.6

```
fun singletons xs = map (fn x => [x]) xs
fun sort xs = mergelists (singletons xs)
```

Question 2 (30 %): Grammar and abstract syntax

Question 2.1

```

Date ::= EVERY Day
      | FIRST Day OF THE MONTH
      | NTH Day OF THE MONTH
      | LAST Day OF THE MONTH
      | NTH LAST Day OF THE MONTH

Day  ::= DAY
      | MONDAY
      | TUESDAY
      | WEDNESDAY
      | THURSDAY
      | FRIDAY
      | SATURDAY
      | SUNDAY
    
```

Question 2.2 and 2.3

```

Main:
    Date EOF                { $1 }
;
Date:
    EVERY Day               { Every $2 }
    | First Day OF THE MONTH { First($1, $2) }
    | Last Day OF THE MONTH  { Last ($1, $2) }
;
Day:
    DAY                     { Day }
    | MONDAY                 { Mon }
    | TUESDAY                 { Tue }
    | WEDNESDAY               { Wed }
    | THURSDAY                 { Thu }
    | FRIDAY                   { Fri }
    | SATURDAY                 { Sat }
    | SUNDAY                   { Sun }
;
First:
    FIRST                     { 1 }
    | NTH                       { $1 }
Last:
    LAST                       { 1 }
    | NTH LAST                   { $1 }
;
    
```

Question 2.4

1st|[2-9]1st|2nd|[2-9]2nd|3rd|[2-9]rd|[4-9]th|[2-9][04-9]th|1[0-9]th

Question 3 (40 %)

Question 3.1

```
CST 100          100
CST 11          11 : 100
ADD             111
MKNIL          Nil : 111
MKCONS         Cons(111, Nil)
PRINT          Cons(111, Nil)
STOP           -
```

The lower half of the resulting drawing is black; the upper half's left half is white and its right half is black. The result is a black square with a white square in the upper left-hand quarter.

Running the program prints:

```
Cons(111, Nil)
```

To create and print the list `Cons(111, Cons(222, Nil))`, use the following program:

```
CST 111, CST 222, MKNIL, MKCONS, MKCONS, PRINT, STOP
```

Question 3.2

```
fun mklist 0 = [MKNIL]
  | mklist n = CST (2 * n) :: mklist (n-1) @ [MKCONS];
val mklist = fn n => mklist n @ [PRINT, STOP];
```

Question 3.3

The program executes as follows:

```
0: CALL 4          42
4: DUP            42 : 42
5: DUP            42 : 42 : 42
6: MKNIL          Nil : 42 : 42 : 42
7: MKCONS         Cons(42, Nil) : 42 : 42
8: MKCONS         Cons(42, Cons(42, Nil)) : 42
9: MKCONS         Cons(42, Cons(42, Cons(42, Nil)))
10: RET           Cons(42, Cons(42, Cons(42, Nil)))
2: PRINT          Cons(42, Cons(42, Cons(42, Nil)))
3: STOP           -
```

Therefore the program prints:

```
Cons(42, Cons(42, Cons(42, Nil)))
```

Question 3.4

```
CALL buildlist, PRINT, STOP
buildlist: DUP, IFNZRO buildcons, POP, MKNIL, RET
buildcons: DUP, CST 1, SUB, CALL buildlist, MKCONS, RET
```

Question 3.5

In the first case, the program prints 111. In the second case, it prints 999.

Question 3.6

```
sum: LISTCASE lab, CST 0, RET,
lab: CALL sum, ADD, RET
```

Question 3.7

The Java implementation of the stack machine needs to be changed in two places. First, the GETVAR instruction must be defined, as in:

```
final static int ..., GETVAR = 13, ...;
```

Second, the switch statement must be extended with a branch that interprets the GETVAR instruction. The program contents $p[pc++]$ after the current instruction address pc is used as an offset from the stack top sp , the result of this subtraction is used to index into the stack, as in $s[sp-p[pc++]]$, and the resulting value is pushed onto the stack top:

```
case GETVAR:
    s[sp+1] = s[sp-p[pc++]];
    sp++;
    break;
```

This defines a function that creates a list containing n copies of v , given a stack of the form $n : v : s$:

```
buildlist: DUP, IFNZRO lab, POP, MKNIL, RET,
lab:      GETVAR 2, SWAP, CST 1, SUB, CALL buildlist, MKCONS, RET
```

Question 3.8

```
CALL buildlist, CALL twicelist, PRINT, STOP
buildlist: DUP, IFNZRO buildcons, POP, MKNIL, RET
buildcons: DUP, CST 1, SUB, CALL buildlist, MKCONS, RET
twicelist: LISTCASE iscons, MKNIL, RET
iscons:    CALL buildcons, DUP, ADD, MKCONS, RET
```