COMPUTER SCIENCE TRIPOS Part IA

Friday 27 May 1994 1.30 to 4.30

Paper 1

Answer **five** *questions.*
No more than **two** *questions from any one section are to be answered.*
Submit the answers in five **separate** *bundles each with its own cover sheet.*
Write on **one** *side of the paper only.*

SECTION A

1 An UP-DOWN binary counter with input `A` is required. If `A`=0 the counter counts up, and if `A`=1 it counts down. Design an implementation for 3 bits using T-type flipflops. [20 marks]

2 Describe the main components that make up a computer. [4 marks]

Illustrate different ways of connecting these components together to span a range of performance requirements. [10 marks]

For each of the performance categories that you identify state today's typical memory size, bus width and CPU speed. [6 marks]

3 Self-timed logic circuits may enable an existing technology to achieve higher performance. Explain the principle of self-timed circuits. [8 marks]

Comment on the possible opportunities for improved performance. [6 marks]

Is it likely that large logic systems will become easier to design using self-timed logic rather than more traditional arrangements? Explain. [6 marks]

4 Explain, in about 25 words each, *five* of the following terms:

(*a*) barrel shifter;

(*b*) depletion layer;

(*c*) state assignment;

(*d*) signal combination with diodes;

(*e*) dynamic logic;

(*f*) binary addition;

(*g*) static hazard. [4 marks each]

SECTION B

5 Study the following ML function definitions and answer the questions below:

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

fun sep [ ]          = [[ ], [ ]]
  | sep [x]          = [[x], [ ]]
  | sep (x::y::rest) = prefix [x,y] (sep rest);

fun merge[[ ],y]         = y : int list
  | merge[x,[ ]]         = x
  | merge[x::xs, y::ys)] =
      if x<y then x :: merge[   xs, y::ys]
             else y :: merge[x::xs,    ys];

fun s [ ] = [ ]
  | s [x] = [x]
  | s  x  = merge (map s (sep x));
```

Deduce the ML type of the function `prefix` and derive the result of the call:

```
prefix [1, 2, 3] [[4], [5], [6]];
```

[2 marks]

Give a correctly-typed call to `prefix` that will generate an exception when evaluated. [2 marks]

What values do `sep[1,2,3,4,5,6,7,8]` and `sep[1,2,3,4,5,6,7]` yield? [4 marks]

Deduce the ML type of `merge` and explain why the omission of '`: int list`' would lead to an error [2 marks]

Give an ML definition of the standard library function `map`. [2 marks]

Describe what the function `s` does and explain why it works. [8 marks]

6 The following ML declaration introduces a data type that can be used to represent a potentially infinite arrangement of cells at positions with integer coordinates in the first quadrant of the $x - y$ plane:

```
datatype A = Z
           | Cell of int ref * A ref * A ref;
```

Each cell contains an integer value and pointers to the cells immediately to its right and above itself. These three components are all mutable so that the arrangement of the cells and the integers they contain can change during use. The constructor `Z` can be used in an `A ref` to allow a cluster to have a boundary rather than continuing through unbounded chains of cells.

Define a function `mkrow(`n`)` of type `int->A` that will return a row of length $n + 1$ cells initialised with zeros. For instance:

```
mkrow(1) = Cell(ref 0, ref(Cell(ref 0, ref Z, ref Z)), ref Z)
```

[5 marks]

Define a function `zip(`*row1*`,` *row2*`)` of type `A*A->A` that will return *row1* with *row2* joined above it. This function is entitled to change some of the `ref A` pointers in *row1*. For example

```
val root = zip(mkrow(3), mkrow(2));
```

would give `root` a value representing the following arrangement.

$$\begin{tikzcd}
 & \texttt{0} \arrow[r] & \texttt{0} \arrow[r] & \texttt{0} & \\
\texttt{root :} & \texttt{0} \arrow[r] \arrow[u] & \texttt{0} \arrow[r] \arrow[u] & \texttt{0} \arrow[r] \arrow[u] & \texttt{0}
\end{tikzcd}$$

[5 marks]

Next define a function `mkarr(`m`,`n`)` of type `(int*int)->A` that will return a value representing a rectangular array of $n + 1$ rows each of which are of length $m + 1$ in which each cell is initialised to zero [5 marks]

Paths originating from the bottom leftmost cell (which will be referred to by the variable `root`) are represented by values of the type `dir list` where `dir` is declared as follows:

```
datatype dir = Right | Up;
```

Finally define a function `inc_path_cells` of type `A->A list->unit` that will increment the integers in all the cells that lie on a specified path within a given collection of cells. For instance after `root` had been set up as above, the two calls

```
inc_path_cells root [Right, Up, Right];
inc_path_cells root [Right, Right, Right];
```

would leave `root` representing the following arrangement:

$$\begin{tikzcd}
 & 0 \arrow[r] & 1 \arrow[r] & 1 & \\
\texttt{root :} & 2 \arrow[u] \arrow[r] & 2 \arrow[u] \arrow[r] & 1 \arrow[u] \arrow[r] & 1
\end{tikzcd}$$

You may assume that the path does not try to reach cells outside the given arrangement. [5 marks]

7 Show, by defining suitable selector and constructor functions, how the ML type defined as follows:

```
datatype L = N of int * unit->L;
```

can be used in the representation of lazy integer lists. [5 marks]

Define a function `makeseq(f)` that will yield a lazy list representing the following infinite sequence:

```
0, f(0), f(f(0)), ...
```

where the integer at position i has the value $\mathtt{f}^i(0)$. [5 marks]

Define a function '`matches s seq`', where `s` is of type `int list` and `seq` is a lazy list, that will yield a lazy list of integers giving the positions where `s` matches consecutive items in `seq`. For example, if `matches [1 1]` is applied to the lazy list

```
1,1,2,1,1,1,0,1,2,1,1,...
```

it will produce a lazy list starting

```
0,3,4,9,...
```

[10 marks]

8 Sets of distinct integers can be implemented in ML as values of type `set` declared below:

```
datatype set = Leaf | N of set * int * set;
```

Describe how you would use this data type to represent sets. [4 marks]

Give simple definitions for the following functions:

(*a*) `insert:  int*set->set`
Returns a set containing the given integer as well as all the elements of the given set; [4 marks]

(*b*) `mkset:  int list->set`
Creates a set containing all the integers from the given list; [3 marks]

(*c*) `mklist:  set->int list`
Makes a list of all the integers present in the given set; [3 marks]

(*d*) `union:  set*set->set`
Forms a set from all integers in the two arguments, avoiding the introduction of repeated entries; [3 marks]

(*e*) `select :  set->int*set`
Returns an arbitrary integer from the set, and also the set with that item removed.
`select` should raise an exception if the given set is empty. [3 marks]

Your definitions should aim for simplicity and elegance rather than efficiency.

SECTION C

9 The Imperial system for Sterling currency was based on the *pound*, the *shilling* and the *penny*, with 12 pence per shilling and 20 shillings per pound. Define a Modula-3 record type to store an amount of money in pounds, shillings and pence. Allow for both positive and negative sums, and use sub-range types to restrict the values of fields in your data structure so that (for instance) the pence field always contains a number in the range 0 to 11. [4 marks]

Write procedures to convert an integer value in pence to the Imperial type you have just defined, and to convert from the Imperial type to text. The following examples illustrate aspects of the desired text corresponding to various numbers of pence. The library procedure `Fmt.Int` may be used to convert integer values to text.

```
   0 →  zero
   1 →  1 penny
  10 →  10 pence
  60 →  5 shillings
  80 →  6 shillings and 8 pence
 252 →  1 pound and 1 shilling
 479 →  1 pound, 19 shillings and 11 pence
1201 →  5 pounds and 1 penny
2400 →  10 pounds
-252 →  minus 1 pound and 1 shilling
```

[16 marks]

Credit will be given for a clearly explained, concise and tidily presented solution. Minor syntax or punctuation errors in the Modula-3 code will not count heavily against you.

10 Modula-3 allows the user to declare arrays with any sort of contents — for instance arrays of integers, reals, `TEXT` or structures, but the index type for an array is restricted and in particular cannot be of type `TEXT`.

It is sometimes useful to achieve an effect analogous to having an array that can be indexed using values of type `TEXT`. One way of doing this is to use a structure known as a *hash table*: given an index value of type `TEXT` an integer is computed using a *hash function* and this is then used to index an array. The library procedure `Text.Hash` computes a suitable integer from a `TEXT` value. Two complications arise. First the integer computed by the hash function may lie outside the valid range of index values for the array. Secondly two different `TEXT` objects may give rise to the same hash value.

The problems can be resolved first by reducing the raw hash value modulo the size of the array and arranging that each array entry refers to the start of a linked list of (*index,value*) pairs. Retrieving a value from the table involves accessing the array to obtain the correct list of pairs and then scanning the list to find an index value that is identical (use the library function `Text.Equal`) with the `TEXT` index being sought. The corresponding value can then be returned. Storing into the table will involve adding a new (*index,value*) pair to one of the lists.

Design appropriate data structures for such a table, and write procedures to store and retrieve values, using the following signatures:

```
PROCEDURE Put(VAR table: Table; key, value: TEXT)
                                 RAISES {DuplicateKey}

PROCEDURE Get(READONLY table: Table; key: TEXT): TEXT
                                 RAISES {MissingKey}
```

[20 marks]

11 In Modula-3 what are object types, and how do they differ from simple record data types?

In this context give a brief explanation of method invocation, inheritance and of each of the keywords `METHODS`, `NEW` and `OVERRIDES`. Include short examples where appropriate. [10 marks]

Describe briefly the facilities in Modula-3 for defining and using array and reference types.

Explain the concept of an *open array*, showing how access to an array that is received as an argument by a procedure may differ from direct references to the same array. Give an example of a programming task that would be harder in Modula-3 if open arrays were not provided. [10 marks]

12 A Unix user (with the BASH shell) sets up a file containing the following commands, and ensures it is executable:

```
echo $1 $2
mv $1 $1.temp
mv $2 $1
mv $1.temp $2
```

The user at the next terminal sets up a file that is very similar, but which uses `cp` rather than `mv`. Describe the behaviour each can expect when they use these files as command scripts. Assuming that the files concerned are both called `sw`, explain carefully the consequences of such uses as

```
sw somefile somefile
sw firstfile.temp secondfile.temp
sw only/one.file
```

[7 marks]

In another file, called `de` (say) the following commands exist:

```
echo $# files >> de.info
for n in $*
  do
    echo $n >> de.info
    mv $n backup/$n
  done
```

What do the various substitutions (involving '$' signs) do in this case? Given that '>>' is much like '>' but appends new data to an existing file rather than creating a new one, what will build up in `de.info` over the course of time? Discuss the effect of issuing the command "`de *`". [7 marks]

Many Unix commands, for example `xlsfonts` and even just `ls`, can generate more output than will fit on the screen at once. Give a brief account of (*a*) how to use `more` to inspect the output and (*b*) how to collect a copy of the output in a file for inspection using a text editor. Write a shell script that will run `ls` with the `-l` flag (to get a full detailed listing of file sizes and dates) on one or more directories, will collect all the output in a single temporary file, enter an editor to allow you to inspect the information you have gathered and at the end get rid of the temporary file. [6 marks]