COMPUTER SCIENCE TRIPOS Part IA

Friday 28 May 1993 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 Describe the operation and gate level circuit of a *multiplexer*. How may it be used in conjunction with a *demultiplexer* to establish alternative connections across a unidirectional path between two modules?

What is involved in extending the method to allow a bidirectional path between two (or more) modules? What do you understand by the terms *bus, master, slave* in connection with your circuit?

2 Design a *divide-by-three* counter using T-type flipflops.

3 Explain the *Synchronization Problem* which affects the operation of certain logic circuits. How should circuits be designed to minimize or avoid the problem?

4 Give a qualitative explanation of current flow across a *p-n junction* in a semiconductor diode. Why is there a depletion layer in the immediate vicinity of the junction?

SECTION B

5 Give an ML definition of the function `map3` which has the property that

$$map3\ f\ [x_1, x_2, \ldots, x_n] = [f\ 0\ x_1\ x_2,\ f\ x_1\ x_2\ x_3, \ldots,\ f\ x_{n-1}\ x_n\ 0]$$

and deduce the type of `map3`. The function `map3iter` is defined as follows:

```
fun map3iter _ (0::_) = 0
 |  map3iter g   x    = 1 + map3iter g (map3 g x);
```

Deduce the type of `map3iter` and explain in words what the function does. Illustrate your answer by considering the call

```
map3iter g [1, 1, 1, 1, 1, 1];
```

in an environment in which `g` is defined as follows:

```
fun g 0 1 _ = 2
 |  g 1 1 _ = 1
 |  g 2 1 _ = 2
 |  g _ 2 0 = 0
 |  g _ n _ = n;
```

6 The structure of a binary tree containing integers at some of its leaves is given by the ML datatype `T` defined as follows:

```
datatype T = X | N of int | D of T*T;
```

Define a function `filter` of type `(int -> bool) -> (T -> T)` with the property that the call `filter p t` will yield a simplified copy of `t` by repeated application of tree rewrite rules:

```
D(X,a) -> a     D(a,X) -> a
```

on the tree obtained from `t` by replacing all leaf nodes of the form `N k` for which `p k` yields `true` by `x`. Thus, for example:

```
filter (fn n => n=0)
```

should yield a function that converts `D(D(N0,N0), D(D(N2,N0), N3))` to `D(N2,N3)`.

7 State carefully what it means to say that a function has time complexity $O(f(n))$, and give ML definitions for some example `int ->int` functions which have time complexities $O(\log n)$, $O(n)$, $O(n^2)$, $O(n^3)$. In what circumstances can a function have time complexity $O(1)$?

Estimate the time complexities of the functions `f1`, `f2` and `f3` defined below:

```
fun f1  0  =  1
 |  f1  n  =  1 + f1(n-1);

fun f2  0  =  1
 |  f2  n  =  f2(n-1) + f1 n;

fun f3  0  =  1
 |  f3  n  =  f3(n div 7) + f3(5*n div 7) + f1 n;
```

8 The structure of a binary tree with integers at the leaves is represented by the following ML datatype:

```
datatype T = n of int | d of T*T;
```

Define a function `flatten` which when given an argument `t` of type `T` will yield the list of integers obtained by a left to right walk over `t`. For example,

```
flatten (d(d(n1,n2),n3)) = [1,2,3].
```

Define a function `splits` which, when given a list of length $n > 0$, will yield the list of 2-tuples representing the $n-1$ ways of splitting the given list into two non-empty sublists. For example,

```
splits [1,2,3] = [ ([1], [2,3]), ([1,2], [3]) ].
```

Hence or otherwise define a function `alltrees` which, when given a list of length $n > 0$, will form a list of all the trees of type `T` that will flatten to the given list. For example,

```
alltrees [1,2,3] = [ d(n1, d(n2, n3)), d(d(n1, n2), n3) ].
```

SECTION C

9 A suite of Modula-3 procedures is being developed to handle arbitrarily large non-negative integers.

A test program for handling such numbers includes the following `TYPE` declaration:

```
TYPE
  Digit = [0..9];
  RefBigNo = REF BigNo;
  BigNo = RECORD dig:Digit:=0;  rest: RefBigNo:= NIL END;
```

A variable of type `RefBigNo` is a reference to an arbitrarily large number, represented as a sequence of base-10 digits stored in reverse order. The chosen data structure is a record which consists of the *last* digit of the number and a reference to the remaining digits.

Write a procedure `Add` which will add two of these large numbers and a procedure `Print` which will convert one such number to type `TEXT`. The following signatures might be appropriate for the two procedures:

```
PROCEDURE Add(a, b: RefBigNo; carry:=0): RefBigNo =
```

and

```
PROCEDURE Print(N:RefBigNo; first:= TRUE): TEXT =
```

10 Distinguish between `TRY-EXCEPT` and `TRY-FINALLY` and provide program fragments to illustrate uses of these Modula-3 clauses.

A Modula-3 program includes the following declarations:

```
TYPE
  ArrCard = ARRAY OF CARDINAL;
  RefArrCard = REF ArrCard;

EXCEPTION
  TooMuch;

PROCEDURE Read(VAR a: RefArrCard) : CARDINAL =
      .
      .
  END Read;

VAR
  buffer:= NEW(RefArrCard, 10);
  count:= Read(buffer);
```

The data for the program are arranged in lines with strictly one `CARDINAL` value on each line so that a loop incorporating `Scan.Int(Rd.GetLine(Stdio.stdin))` will input successive values.

As indicated, the variable `buffer` initially refers to an array of 10 `CARDINAL`s and this is handed to the procedure `Read` whose task is to read values from the data and assign them to successive elements of the array.

Should there be more values in the data than can be accommodated in the array, the procedure `Read` will raise the `TooMuch` exception. The exception handler will create a new array which is larger by 20%, transfer values from the full array to the new one, and then continue reading from the data. This enlargement process will be repeated as necessary until all the data have been read. It may be assumed that there is unlimited memory.

When the reading is complete, `buffer` will refer to the most recently created array and the procedure `read` will return, for assignment to the variable `count`, the number of values that have been read from the data.

Complete the procedure `Read`.

11 What is a hierarchical file structure? Discuss the good and bad points about such a file structure.

Give an example of the use of a hierarchical file structure in a UNIX system by presenting a diagram, and use your example to describe the following terms:

(a) home directory;

(b) current directory;

(c) full pathname;

(d) relative pathname.

Imagine that you have created a UNIX file structure in which your leaf nodes are very deep. How can you arrange to get to specific leaf nodes quickly?

12 A list in Modula-3 can be represented as a sequence of links. Each link is a record containing one value in the list and a reference to the rest of the list following the link. The following `TYPE` declaration specifies the required data structure:

```
TYPE
  List = REF Link;
  Link = RECORD value:CARDINAL; rest: List:= NIL END;
```

A test program which exploits lists of this kind includes:

```
VAR
  start:List;

BEGIN
  start:= NIL;
  Put(10,start);
  Put(100,start);
  Put(1000,start);
  Print(start);
  Print(Reverse1(start));
  Print(reverse2(start));
```

The procedure call `Put(1000,start)` will add a link containing the value `1000` to the *end* of the list which already includes the values `10` and `100`.

The procedure `Print` writes out the values in a list in order.

The procedures `Reverse1` and `Reverse2` reverse a list in two different ways, equivalent to the ML functions:

```
fun Reverse1 [] = []
 |  Reverse1 (value::rest) = Reverse1 (rest) @ [value];

fun Reverse2 list =
  let
    fun rev ([],result) = result
     |  rev (value::rest, result) = rev (rest,value::result)
  in
    rev (list,[])
  end;
```

Write the Modula-3 procedures `Put, Print, Reverse1` and `Reverse2`.