UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1997

BEng Honours Degree in Computing Part II
MEng Honours Degrees in Computing Part II
BSc Honours Degree in Mathematics and Computer Science Part II
MSci Honours Degree in Mathematics and Computer Science Part II
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the
Associateship of the Royal College of Science
Associateship of the City and Guilds of London Institute

PAPER 2.3 / MC2.3

COMPILERS

Monday, April 28th 1997, 4.00 - 5.30

Answer THREE questions

For admin. only: paper contains 4 questions

1 Consider the following grammar:

deflist	→ 'define' def | deflist def
def	→ namelist '=' namelist
namelist	→ namelist name | name

a **Modify this grammar to make it suitable for direct construction of a recursive descent parser.**

b **Using Miranda or another convenient programming language, write a recursive descent parser for this language. Your parser need not construct an abstract syntax tree, but it should generate an error message if a syntax error is found.**

Assume that lexical tokens are represented by the following Miranda data type:

```
token ::= Define | Eqsign | Name [char]
```

c Suppose the language is modified as follows:

deflist	→ 'define' def | deflist def
def	→ namelist '=' exp
namelist	→ namelist name | name
exp	→ namelist | exp 'where' deflist

Are there any difficulties in writing a recursive-descent parser for this language? Give an example which illustrates what the problem is.

d Explain what the problem is using a formal analysis of the grammar. Explain carefully what your analysis shows.

(The four parts carry, respectively, 20%, 40%, 10%, and 30% of the marks).

2 This question concerns "for" loops in a simple programming language. In Miranda, a suitable (simplified) abstract syntax tree data type is as follows:

```
stat ::= For name num exp stat | Assign name exp
exp ::= Const num | Var name | Divide exp exp
```

The target machine's assembly code is represented in Miranda as follows:

```
instruction ::= Define label |     -- assembler directive to define label
                Jmp label |        -- unconditional jump to label
                Add reg reg |      -- Add r1 r2 means r1 := r1 + r2
                Sub reg reg |      -- Sub r1 r2 means r1 := r1 - r2
                Div reg reg |      -- Div r1 r2 means r1 := r1 / r2
                Load reg name |    -- Load r1 x means load variable named x into r1
                LoadI reg num |    -- LoadI r1 n means constant n into r1
                Store reg name |   -- Store r1 x means store r1 into variable named x
                Bgtz reg label |   -- Branch to label if reg > 0
                Bltz reg label |   -- Branch to label if reg < 0
                Bgez reg label |   -- Branch to label if reg >= 0
                Blez reg label     -- Branch to label if reg <= 0

reg ::= R0 | R1 | .. | R31
```

Assume that you have already been provided with a code generation function `transExp` for expressions. It uses only those registers given by its second parameter, and it leaves the result in the first register in the list.

a Consider the following loop, which tests that `x` is prime, using as few iterations as possible:

```
S := x div 2
for i : 2 .. S
  if x mod i = 0 then put "not prime" end if
  S := (S + x div S) div 2
end for
```

Should your code generator give the programmer's intended behaviour for this example? Explain.

b Use Miranda to sketch a simple code generator for statements (i.e. for `stat` above). Your code generator *should* give the intended behaviour for the example above. Make any assumptions necessary, but take care to state them in your answer.

c Consider the following alternative implementation of the example loop above:

```
for i : 2 .. isqrt(x)
  if x mod i = 0 then put "not prime" end if
end for
```

(the built-in function `isqrt()` finds the nearest integer to the square root of `x`). Show clearly how your code generation for `stat` would have to be modified, so that evaluation of the expression `isqrt(x)` is not repeated unnecessarily. You do not have to write out all the details; full marks will be awarded for a clear explanation.

(The four parts carry, respectively, 20%, 40%, and 40% of the marks).

Turn over ...

3 This question concerns code generation for a simple programming language. The target machine's assembly code is represented in Miranda as follows:

```
instruction ::= Define label |     -- assembler directive to define label
                Jmp label |        -- unconditional jump to label
                Div reg reg |      -- Div r1 r2 means r1 := r1 / r2
                Load reg name |    -- Load r1 x means load variable named x into r1
                LoadI reg num |    -- LoadI r1 n means constant n into r1
                Store reg name |   -- Store r1 x means store r1 into variable named x
                Jsr name |         -- push PC onto stack and jump to label given
                Ret                -- pop address from top of stack and jump to it

reg ::= R0 | R1 | .. | R31
```

a This part of the question concerns expressions. In Miranda, a suitable (simplified) abstract syntax tree data type is as follows:

```
exp ::= Const num | Var name | Divide exp exp
```

Use Miranda to sketch a simple code generator for expressions (i.e. for **exp** above). Your translation function should use registers efficiently, and should use only registers from a specified list.

b This part of the question concerns procedure declarations and procedure calls. In Miranda, a suitable (simplified) abstract syntax tree data type for statements is as follows:

```
stat ::= Assign name exp | Call name paramList
paramList == [exp]
```

(Note that the notation `[exp]` represents a *list* of expressions, the parameters for the procedure). The abstract syntax tree for procedure declarations is as follows:

```
procDecl ::= Proc name paramDeclList Body
paramDeclList == [name]
Body == [stat]
```

Here, the list of statements `[stat]` constitutes the body of the procedure. For simplicity, we assume all parameters are integers, there are no local variables, and no nested procedure declarations.

In this machine, parameters are always passed in registers, starting with register `R0`, `R1` and so on.

i Use Miranda to *sketch* a simple code generator for procedure calls.

ii Use Miranda to *sketch briefly* a simple code generator for procedure declarations. Explain carefully any assumptions you have to make.

(The three parts carry, respectively, 40%, 20%, and 40% of the marks).

4 Consider the following fragment of Turing code:

```
type Node :
  record
    ident : int
    NoOfNbours : int
    Nbours : array 1..3 of pointer to Node
  end record

proc Connect(n1, n2 : pointer to Node)
  n1->NoOfNbours := n1->NoOfNbours + 1
  n2->NoOfNbours := n2->NoOfNbours + 1
  n1->Nbours(n1->NoOfNbours) := n2
  n2->Nbours(n2->NoOfNbours) := n1
end Connect

function Dtree(d : int) : pointer to Node
  var n, nL, nR : pointer to Node
  if d<=1 then
    new n
    n->ident := d
    n->NoOfNbours := 0
    result n
  else
    nL := Dtree(d-1)
    nR := Dtree(d-2)
    Connect(nL, nR)
    result nL
  end if
end Dtree
```

a Suppose the following function call is executed:

```
tree = DTree(3);
```

Draw a sequence of diagrams showing the state of the stack as it appears each time `DTree` is entered. Show all variables, parameters, results, return addresses and links, but *do not* show the `Nodes`, and do not show the operation of the procedure `Connect`.

b Explain how garbage collection can be achieved using the two-space copying algorithm.

c In the example above, each `Node` contains an array of pointers. Suppose a copying garbage collector is called at a point where the array has not yet been initialised, so contains random data. What problem would arise?

d Suggest an alternative garbage collection scheme, and explain how it would overcome this problem.

(The four parts carry, respectively, 50%, 25%, 10% and 15% of the marks).