UNIVERSITY OF LONDON

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1999

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 / MC 2.3

COMPILERS I

Friday, April 30th 1999, 10.00 – 11.30

Answer THREE questions

For admin. only:
paper contains 4 questions

1 Consider the following expression language:

```
exp ::= exp - exp | const | ident | (exp)
```

(with - being left associative).

a Using tokens:

```
tokens ::= Sub | Num num | Ident string | Lbrac | Rbrac
```

and abstract syntax tree:

```
tree ::= Const num | Var string | Minus tree tree
```

write a recursive-descent parser for a modified version of the grammar, using Miranda or another convenient programming language.

b Using Miranda or another convenient programming language, write down a function which assigns Sethi-Ullman numbers to expressions. Ilustrate the working of the function using the following example:

```
Minus(Minus(Const 3)(Const 2))(Minus(Const 5)(Const 6))
```

c Using the following target machine assembly code:

```
instruction ::=    Subtract reg reg | -- means r1 := r1-r2
                   Load reg name |
                   LoadI reg num
reg ::= R0 | R1 | ... | R31
```

sketch a simple code generator for expressions (using Miranda or another convenient programming language). Your translation function should use registers efficiently, and should use only registers from a specified list.

2a Sketch a diagram showing the main phases of a compiler.

b In a compiler for a simple programming language, Boolean expressions are represented using the following abstract syntax:

```
bexp ::=   True | False | Or bexp bexp | And bexp bexp |
           Var name | Not bexp
```

Using Miranda or another convenient programming language, write a function which generates code to test the value of a supplied Boolean expression and branches to one of two labels (supplied as parameters of type `name`) as soon as the expression is known to be `True`, and to the other as soon as it is known to be `False`.

Boolean variables are represented at run time using integers with the value 0 for false and 1 for true. Your machine has registers and instructions `BEQZ reg lab` (branch if register `reg` is zero) and `BNEQZ reg lab` (branch if register `reg` is non-zero) for testing their values.

You may find it useful to assume the existence of a "function" `newlabel`, which generates a fresh label not used elsewhere.

c A simplified abstract syntax for statements is:

```
stat ::= Repeat stat bexp | Assign name bexp
```

where the first alternative is a repeat loop (execute the statement until the boolean becomes true) and the second is an assignment to a boolean variable. Use Miranda or another convenient programming language to sketch a simple code generator for statements. You may invent your own instructions but you should be sure to specify the intended meaning of any instructions that you use.

(The three parts carry, respectively, 25%, 50% and 25% of the marks)

Turn Over ...

3 Consider the following grammar for a fragment of a programming language:

```
S ::=   if b then S else S |
        if b then S |
        S ; S |
        s
```

a Modify the grammar to yield a definition which is suitable for recursive-descent parsing.

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

```
token ::= Bool | Stat | Semi | If | Then | Else
```

Using Miranda or another convenient programming language, sketch a recursive descent parser for this language. Your parser need not construct an abstract syntax tree.

c Show, by computing appropriate FIRST and FOLLOW sets, that this grammar has a context clash.

d Give an example which illustrates the ambiguity of the grammar and suggest how the language might be modified to overcome this problem.

4a 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
```

i) Explain what information will have to be maintained in the symbol table so that the compiler can generate correct code and check for errors.

ii) Describe what is involved in typechecking the final statement of `Connect`

b Consider the following procedure:

```
BEGIN
    VAR arg : INTEGER;
    PROC fac(n : INTEGER):INTEGER;
        BEGIN
            IF n <= 0 THEN RETURN 1
            ELSE RETURN (n * fac(n-1))
            ENDIF;
        END;
    arg := 3; print(fac(arg));
END
```

Write down a sequence of snapshots of the stack prior to each call of the procedure. Be careful to show the static and dynamic links for each frame.