UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2000

BEng Honours Degree in Computing Part II
MEng Honours Degrees in Computing Part II
BEng Honours Degree in Mathematics and Computer Science Part II
MEng 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 City and Guilds of London Institute
This paper is also taken for the relevant examinations for the Associateship of the Royal College of Science

PAPER C221=MC221

COMPILERS

Thursday 4 May 2000, 14:00
Duration: 90 minutes
(Reading time 5 minutes)

Answer THREE questions

Paper contains 4 questions

1a Using the following definition of abstract syntax trees:

```
data Tree = Const Int | Var String |
            Add Tree Tree | Times Tree Tree
```

write down a function which assigns *Sethi-Ullman* numbers to expressions, using Haskell or another convenient programming language. Illustrate the working of the function using the following example:

```
Add (Const 3) (Times (Const 6) (Var "x"))
```

b Using the following target machine assembly code:

```
data Inst = Plus Reg Reg |      -- means r1 := r1+r2
            Times Reg Reg |     -- similarly
            Load Reg String |   -- r := String
            LoadI Reg Int       -- r := Int

data Reg = R0 | R1 | ... | R31
```

Sketch how the Sethi-Ullman numbers may be used to optimise the use of registers in the translation of expressions (your translation function, of type Tree -> [Reg] -> [Inst], should use only registers from a specified list).

c i) Sketch the basic principles of the *graph colouring* approach to register allocation and describe why it is superior to the Sethi-Ullman numbering scheme.

ii) Construct the interference graph for the following program:

```
A:=e1; B:=e2; ... B used ... C:=5; ... D:=A+6;
... D used ... C used ...
```

iii) Suggest a colouring for the nodes in the interference graph.

2 Consider the following grammar for a fragment of the English language:

```
S ::=   NP Verb {PP}
NP ::=  Noun {PP}
PP ::= 'on' NP
Verb ::= 'sleep' | 'appear'
Noun ::= 'ideas' | 'boats' | 'Tuesdays'
```

a Modify the grammar to eliminate the {_} notation.

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

```
data Token = On | Sleep | Appear | Ideas | Boats |
             Tuesdays
```

Using Haskell or another convenient programming language, sketch 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.

c Show, by computing FIRST and FOLLOW sets for all non-terminals, that this grammar has a context clash.

d Show that this grammar is ambiguous by giving an example.

(The four parts carry, respectively, 10%, 45%, 35% and 10% of the marks)

3a *Consider the following conditional statement:*

```
if a and not(b or c) then d := 1 endif
```

Briefly explain the use of short-circuiting in the translation of Boolean expressions (you are not expected to give translation schemes). Give two different reasons why short-circuiting of Boolean expressions might be inefficient.

b A simplified abstract syntax for a new loop construct is:

```
data Stat = Times Exp Stat | ... as standard ...
```

The loop is a Times loop which executes the statement (the body of the loop) the number of times specified by the expression. Use Haskell or another convenient programming language to sketch a simple code generator for the new loop construct. You may assume that the abstract machine has a fixed set of registers and a stack and that a suitable translator for expressions exists; you may invent your own instructions but you should be sure to specify the intended meaning of any instructions that you use.

c Explain the lexical (static) scope rule and the dynamic scope rule for accessing variables. Construct an example and use it for illustrating the difference between the two rules.

4a Explain, with suitable examples, what *Peephole Optimisation* is.

b Describe what information *Reaching Definitions Analysis* collects. Write down the Reaching Definitions equations for a suitably labelled version of the following program:

```
z := 1; while x > 0 do (z := z * y; x := x - 1;)
```

Solve the equations. Explain how Reaching Definitions Analysis as a basis for a *Constant Folding* transformation.

c Consider the following Turing type declaration:

```
type Event:
     record
        time : int
        duration : int
        NoOfSubevents : int
        Subevents : array 1..5 of pointer to Event
     end record
```

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