UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2001

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
BSc Honours Degree in Mathematics and Computer Science Part III
MSci Honours Degree in Mathematics and Computer Science Part III
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

Wednesday 9 May 2001, 14:00
Duration: 90 minutes
(Reading time 5 minutes)

Answer THREE questions

Paper contains 4 questions
Calculators not required

1a Explain briefly (100 words each) the functions of each of the following components of a language-processing system:

i. Lexical Analyser

ii. Syntax Analyser (Parser)

iii Semantic Analyser (Type-Checker)

iv. Evaluator

v. Decompiler

vi Code Generator

b State with examples what data structures are used for input, output, and internal purposes by each of the components in *a* above.

c Draw diagrams showing how the components of *a* above can be combined together into:

i. A Compiler

ii. An Interpreter

iii A Source-to-Source Translator

In each case you should indicate clearly which of the data structures of *b* above are used to communicate between the components.

The three parts carry, respectively, 45%, 35%, 30% of the marks.

2 In a fragment of a language, the abstract syntax trees for expressions are defined by the following Haskell data type:

```
data Exp = Plus  Exp  Exp
         | Var   Name
         | Const Int
         | Index Name Exp
```

For example, the expression `x + 1` is represented as:

```
Plus ( Var "x" ) ( Const 1 )
```

The `Index` node is used to reference elements of one-dimensional arrays; thus the expression `A ( x + 1)` is represented as:

```
Index "A" ( Plus ( Var "x" ) ( Const 1 ) )
```

a Use Haskell to sketch the design of a simple code generator for expressions represented using this data type. The output from your code generator should be for a *zero-address* (stack) machine. State *clearly* any assumptions you make about the target instruction set.

b Now consider the same expression language augmented with function calls taking *a single* parameter. The modified abstract syntax tree is as follows:

```
data Exp = Plus  Exp  Exp
         | Var   Name
         | Const Int
         | Index Name Exp
         | Call  Name Exp
```

Show how your code generator would be extended to handle this.

The two parts carry, respectively, 50%, 50% of the marks.

3 A language provides expressions which consist of integer and Boolean *constants*, a dyadic *addition* operation to add two integer values, a dyadic *comparison* operation to compare two operands of the same type for equality (yielding a Boolean result), and a triadic *choice* operation to evaluate one of two expressions of the same type according to the value of a Boolean expression.

a i Suggest an appropriate Haskell data type for representing expressions in the language as *Abstract Syntax Trees (ASTs).*

ii *W*rite an *Evaluator* to find the *value* of a semantically correct expression (*i.e.* having operands of the correct types) represented as an AST.

b i Suggest an appropriate Haskell data type for representing the *type* of a value in the language.

ii Write a *Type-Checker* to find the *type* of an expression represented as an AST, or to report that it is semantically incorrect (it need not report the reason for the error).

The two parts carry, respectively, 55%, 45% of the marks.

4 A functional language has the following partial grammar for expressions:

```
<exp>  ::= <var> | <cond>
<cond> ::= <exp> if <exp> else <exp>
```

a Explain why the grammar above is unsuitable for *top-down* parsing and transform it Into a version which is suitable.

b Explain why the original version of the grammar is *ambiguous*, justifying your explanation by showing a parse tree of an ambiguous sentence or by formally manipulating the grammar.

c State whether your modified grammar from part *a* is ambiguous and justify your answer.

d Explain a strategy which can be used in a top-down parser to overcome such ambiguities.

e Suggest how the *concrete syntax* of conditional expressions in the language might be redesigned to remove the original ambiguity,

The five parts carry, respectively, 20%, 25%, 30%, 10%, 15% of the marks.