1. (a) Parser-generators such as Yacc, Bison and CUP all provide the compiler writer with commands for specifying the *precedence* and *associativity* (or *non-associativity*) of operators in a programming language. Explain the difference between precedence and associativity and show how both are specified in any one of the parser-generators listed above.

[8 marks]

(b) In keeping with the tradition in mathematical notation, most programming languages allow the same symbol to denote both the subtraction operator (in $x - y$) and the negation operator (in $-x$). Make clear the difficulty which this causes for a parser-generator and explain how it may be overcome.

[5 marks]

(c) A modification to the Tiger language is being considered which will allow logical expressions in Tiger to include both the short-circuiting Boolean operators (written `&` and `|`) and versions of these which always evaluate both of their sub-expressions (written `&&` and `||`).

(i) Give a grammar for these logical expressions omitting the semantic actions for construction of the abstract syntax representation of the input. Include specifications of precedence and associativity. You should state your answer in the form of the input to the Yacc/Bison parser generator *or* in the form of the input to the CUP parser generator.

[6 marks]

(ii) The two short-circuiting boolean operators can be regarded as abbreviations for other expressions. They can be removed by a parser and replaced by the equivalent expression. The two operators which do not perform short-circuiting evaluation can be treated in this way also. Explain how, in all four cases.

[6 marks]

2. (a) Explain the difference between *callee-save* and *caller-save* registers. Why might caller-save registers sometimes not be saved?

   [7 marks]

   (b) Variables which are introduced in a program can occasionally be removed by the compiler. Why is this so? Give examples of programs where variables could be removed and explain how in each case.

   [9 marks]

   (c) In certain cases a compiler writer might decide that it is profitable to perform *in-line expansion* on a function. This is the operation of replacing every call to the function with a modified copy of the function body with the formal parameters replaced with the actual parameters throughout. For example, the program

```
let  function double (x : int) : int = x + x
in   double (13)
end
```

   would become the expression `13 + 13`. However, the transformation is not always applicable and will sometimes result in a slower program.

   (i) What are the circumstances in which the version of the program after expansion is likely to be slower than the original?

   [3 marks]

   (ii) What are the circumstances in which the version of the program after expansion is likely to produce different results from the original?

   [3 marks]

   (iii) For what kinds of functions does this transformation fail to produce a syntactically correct program?

   [3 marks]

3. (a) In the *instruction selection* phase of a compiler the intention is to *tile* an intermediate representation tree with non-overlapping tree patterns which correspond to legal machine instructions? What is an *optimum* tiling? What is an *optimal* tiling?

[6 marks]

(b) Explain the *Maximal Munch* algorithm for the tiling problem. Does it produce an optimum tiling or an optimal one?

[5 marks]

(c) Explain the dynamic programming algorithm for the tiling problem. Does it produce an optimum tiling or an optimal one?

[5 marks]

(d) Give an example of an instruction which could be in the intermediate representation language but would be unlikely to be provided by a real machine.

[3 marks]

(e) Show how the intermediate representation of the Tiger language expression `a[0] := 0` might be tiled.

[6 marks]

Please note the following important points about this specimen paper.

- This paper has not been subjected to the usual rigorous scrutiny applied to all University of Edinburgh examination papers.
- The presence of a question on this paper does not indicate that a question on a similar topic will appear on a future Compiling Techniques examination. A similar question might appear, or it might not.
- In keeping with Departmental policy, specimen answers to this and other examination papers will not be made available.

Stephen Gilmore.
May 4, 1998.