1. (a) What programming language feature leads to the need for an implementation model that includes stack frames? Explain your answer. Explain in detail how a stack frame is pushed to the stack, and removed from the stack, during program execution. [25]

   (b) Comment on how local variables, arguments and non-local variables are addressed in a stack frame by the code generated for a procedure or method. [15]

   (c) Suppose that a compiler translates a MiniJava-like language to an intermediate representation (for example IR trees) that will include the calculations required to address variables in stack frames. Draw or write down the intermediate representation required to access a local variable declared in a method. Explain your answer. [25]

   (d) Explain why registers might be used for parameter passing and suggest situations where passing in registers is particularly appropriate. Outline situations where it is necessary for the code generated for a procedure or method to write registers to the stack. [20]

   (e) Explain the difference between *caller-save* and *callee-save* registers. Why might caller-save registers sometimes not be saved? [15]

2. The reference manual for a MiniJava-like programming language contains the following grammar fragment for an expression involving the `?:` (short-cut if-else) operator:

$$Exp \rightarrow Exp \ ? \ Exp \ : \ Exp$$

We will call this expression the short-cut if-else expression. This expression has the same meaning and type-checks as expressions involving the short-cut if-else operator in Java – `op1?op2:op3` returns `op2` if `op1` is true, and returns `op3` if `op1` is false.

(a) Sketch a possible abstract syntax for the short-cut if-else expression. [15]

(b) Show how semantic actions in a grammar for a parser-generator such as JavaCC can be used to produce abstract syntax trees for the short-cut if-else expression [30]

(c) Informally describe an appropriate typecheck for the short-cut if-else expression. [20]

(d) Suppose a compiler for a MiniJava-like language that includes a short-cut if-else expression translates all statements and expressions into intermediate code (eg intermediate representation (IR) trees). Outline the intermediate code that might be generated in translation of the short-cut if-else expression. You may wish to use a simple example to explain your translation, eg:

```
c = a < b ? a + b : a - b ;
```

You can assume that the expression tree for any variable `v` is simply `TEMP v`. [35]

3. (a) The following regular expression recognises certain strings consisting of the letters a, b and c:

$$a(b|(bc))*c*$$

Indicate which of these five strings are recognised by the above regular expression:

acc, $abac$, a, $abcbcbccc$, $abbbccbc$

Also, show three more strings that are recognised by the above expression. Finally, show two more strings consisting of the letters a, b and c that are *not* recognised by the above regular expression. [25]

(b) Consider the following grammar, which we will call *E*:

E	→	*E* + *E*
E	→	*E* - *E*
E	→	(*E*)
E	→	`num`

i. Explain what it means for a context-free grammar to be ambiguous. Show that grammar *E* is ambiguous. [20]

ii. Explain why grammar *E* is not suitable to form the basis for a recursive descent parser. [10]

iii. Rewrite the rules to obtain an equivalent grammar which can be used as the basis for a recursive descent parser. [25]

(c) Consider the following Java method:

```
1  class A {
2   String a; int c;
3   public void f(int b, String c) {
4     System.out.println(c);
5     int d = 3;
6     int a = b;
7     System.out.println(a+d); System.out.println(b);
8     System.out.println(c); System.out.println(d);
9   }
10 }
```

Given an initial environment σ_0, derive the type binding environments for the method at each use of an identifier and indicate where type lookups will occur. [20]