1. The reference manual for a MiniJava-like programming language contains the following grammar for a for-statement:

$$\textit{Statement} \rightarrow \textbf{for}\ (\textit{id} = \textit{Exp}\ ;\ \textit{Exp}\ ;\ \textit{id} = \textit{Exp})\ \textit{Statement}$$

   (a) Sketch a possible abstract syntax for the for-statement. [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 for-statement. [25]

   (c) Informally describe an appropriate typecheck for the for-statement. [10]

   (d) Suppose a compiler for a MiniJava-like language that includes a for-statement translates all statements and expressions into intermediate code (eg intermediate representation (IR) trees).

      i. Draw or write down the intermediate representation required to access a local variable declared in a method. Explain your answer. [15]

      ii. Outline the intermediate code that might be generated in translation of the for-statement. You may wish to use a simple example to explain your translation, eg:

```
for (i = j; i < k; i=i+1)
   { x = i*i; System.out.println (x); }
```

      You can assume that the expression tree for any variable `v` is simply `TEMP v`. Do not show translations for the body of the example for-statement (in braces in this example `{...}`). [35]

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

$$a((ab)|c) * c$$

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

acc, $abac$, ac, $abababababacac$, $aabcc$

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. [30]

(b) Explain what it means for a context-free grammar to be ambiguous. Write down an ambiguous grammar and show why it is ambiguous. [20]

(c) Consider the following grammar:

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

i. Explain why this grammar is not suitable to form the basis for a recursive descent parser. [10]

ii. Rewrite the rules to obtain an equivalent grammar which can be used as the basis for a recursive descent parser. [20]

(d) Consider the following Java method:

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

3. (a) Choose a programming language you know well and describe how run-time storage is organised and managed during program execution. Clearly associate any storage structures you mention with the implementation of particular language features. [25]

   (b) What is a stack frame? Outline a typical layout for a stack frame and describe each element of a frame and how it is pushed to the stack during program execution. Comment on how local variables, arguments and non-local variables are addressed by the code generated for a procedure or method. [25]

   (c) 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. [30]

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

   [20]