1. (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]

2 marks for each line

2. 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. [20]

(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. [30]

(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).

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 `{...}`). [40]

3. (a) Why do many programming language implementations require a memory model that implements a runtime stack? Explain in detail how a stack frame is pushed to the stack, and removed from the stack, during program execution. [30]

(b) Some programming language implementations avoid in some circumstances the need to pass parameters via a stack frame. Outline what these circumstances might be and why passing via the stack frame might be avoided. Also, outline situations where the use of a stack frame to pass parameters cannot usually be avoided. [25]

(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. [20]

(d) Explain the difference between *caller-save* and *callee-save* registers. Study the following methods and suggest for each whether a caller-save or callee-save register is appropriate for variable `x`. Explain your answers.

```
int f (int a) { int x; x=a+1; g(x); return x+2; }

void p (int y) { int x; x=y+q(y); q(2); q(y+1)}
```

[25]