Question 1.

(a) Describe what is meant by code generation in contrast to compilation. [4 marks]

(b) Explain the difference between the transformative and generative code generation paradigms and give example technologies of each type. [4 marks]

(c) Explain what is meant by horizontal and vertical transformations in code generation. [4 marks]

(d) Why do compositional generators typically only apply to vertical transformations? [3 marks]
Question 2.

(a) Describe the three different kinds of hygiene ascribed to C style macros. [5 marks]

(b) Although the C/C++ programming languages support multi-dimensional arrays, it is more efficient to use a single dimensional array. Write C style macros to implement the following using only a single dimensional array:

- \texttt{MAKE2DARRAY(T,N,M,a)} declares a 2 dimensional array with entries of type $T$, the number of rows and columns is $N$ and $M$ respectively, and the name of the array is $a$.
- \texttt{SET2D(a,i,j,v)} writes the value $v$ in to the 2 dimensional array named $a$ at position $(i, j)$
- \texttt{GET2D(a,i,j)} gives the value of the $(i, j)^{th}$ entry of the 2 dimensional array $a$. [5 marks]

(c) Without explicitly declaring an array, use your macros from (b) above to implement the following C style macros:

- \texttt{MAKE3DARRAY(T,N,M,K,a)} declares a 3 dimensional array with entries of type $T$, with size $N \times M \times K$, and the name of the array is $a$.
- \texttt{SET3D(a,i,j,k,v)} writes the value $v$ in to 3 dimensional array named $a$ at position $(i, j, k)$
- \texttt{GET3D(a,i,j,k)} gives the value of the $(i, j, k)^{th}$ entry of the 3 dimensional array $a$. [3 marks]

(d) Explain in what way the macros above may fail to be hygienic. [2 marks]
Question 3.

(a) Explain how the \texttt{cflow} and \texttt{cflowbelow} pointcut designators affect the semantics of a pointcut in the AspectJ programming language.  

\hspace*{1cm} [2 marks]

(b) Describe the joinpoints matched by the following pointcut designators:

(i) \texttt{call(void java.awt..Component.set*(int,int))}

(ii) \texttt{call(!private static * Integer.*(*))}

(iii) \texttt{call(void *(int, .. ))}

\hspace*{1cm} [6 marks]

(c) We say that a method is \textit{unary} if it accepts a single argument of reference type and returns a reference type. We call a unary method \textit{strict} if, whenever it is passed a null reference as an argument, then it returns a null reference.

Write an AspectJ program that transforms all calls to all public unary methods in to calls to strict versions of those methods.

\hspace*{1cm} [4 marks]

(d) Modify your AspectJ code in (c) above so that only the top-level, outermost calls to unary methods are enforced to be strict. That is, calls to unary methods from within another call to a unary method are allowed to be non-strict.

\hspace*{1cm} [3 marks]
Question 4.

(a) Explain the quote/unquotation mechanism in Meta-AspectJ with particular reference to the Unquote Splice operation ( # a ) where a is an array of AST elements.

[6 marks]

(b) Suppose we wish to generate a Java class called Mins that provides a static min method for calculating the minimum of a number of int values. Given a maximum number of arguments, N, we can use Meta-AspectJ to generate a whole suite of methods that support input of up to N arguments. Note that, \( \min(a_0, a_1, \ldots, a_k) \) is the minimum of \( a_k \) and the result of \( \min(a_0, a_1, \ldots, a_{k-1}) \) and that \( \min(a_0) = a_0 \). The Meta-AspectJ code below is a partial implementation of this generator:

```java
public static void main(String myargs[]) {
    final int MAX_NO_ARGS = 10;

    FormalDec[] params = new FormalDec[MAX_NO_ARGS];
    JavaExpr[] as = new JavaExpr[MAX_NO_ARGS];
    MethodDec[] methds = new MethodDec[MAX_NO_ARGS+1];

    for (int I=0; I<MAX_NO_ARGS; I++){
        params[I] = ... // Each entry should be a parameter
                      // declaration ‘int argI’ where
                      // I ranges from 0 to MAX_NO_ARGS
        as[I] = ... // Each entry should be an expression
                      // of access of the parameter ‘argI’
    }

    methds[0] = ‘[ public int min(int i0){ return i0; }];
```
for(int I=1; I< MAX_NO_ARGS-1; I++){
    FormalDec[] fda = new FormalDec[I+1];
    JavaExpr[] je = new JavaExpr[I];
    System.arraycopy(params, 0, fda, 0, I+1);
    System.arraycopy(as, 0, je, 0, I);
    JavaExpr lastarg = as[I];

    methds[I] = ...
    // Should be a definition of a public static method
    // int min(int arg0, int arg1, ... , int argI+1)

    infer cls = ...
    // Should be a definition of a public class Mins
    // featuring all of the static min methods.

    System.out.println(cls.unparse());
}

[Note that System.arraycopy(src, pos, des, pos’, len) is a
standard Java library call that copies the contents of array src from
position pos in to array des from position pos’ for len entries of the
array.]

Complete this code with Meta-AspectJ commands to provide a gen-
erator for class Mins.

[9 marks]
Question 5.

(a) Describe the instantiation model for C++ templates and explain what is meant by *template specialisation*.

[5 marks]

(b) Under what circumstances do we get a compile-time error due to specialisation? Give an example to demonstrate each circumstance.

[4 marks]

(c) Write a C++ template meta-program called LOOP that, when instantiated with an integer value $N$, say it will print “Start” followed on a new line by a countdown from $N$ to 0. It will then print a new line followed by a count up from 0 to $N$ followed by a new line, “End” and a final new line. An example of output for LOOP instantiated with 7 is below:

```
Start
Down:0 Down:1 Down:2 Down:3 Down:4 Down:5 Down:6 Down:7
End
```

[6 marks]