Answer four out of five questions.

Each question is worth 15 marks.
This examination is worth 60% of the overall module mark.
The coursework is worth 40% of the overall module mark.

Calculators may NOT be used.
1. a) List three benefits of automatic code generation, and explain why code generators provide these. (3 marks)

b) Name three different code generation paradigms. (3 marks)

c) List three reasons why reuse based on libraries and components has reached its limits. Explain how code generation techniques can overcome these limitations. (6 marks)

d) Explain what a generator framework is. What advantages do they provide for building code generators? (3 marks)

2. a) Explain when a macro is called hygienic. List the different levels of macro hygiene that exist, and explain what assurance they provide, and how this can be achieved. (6 marks)

b) Consider the following clone-method, which creates a Point-object and initializes it to the CPoint-object given as argument:

```cpp
Point cloneCPoint(CPoint cp) {
    Point p = new Point;
    p = (Point) cp;
    return p;
}
```

i) Write a C-style macro MK_CLONE that generates a clone-method for two given arguments. Make sure that users can use your macro to generate multiple clone-methods within one class. (5 marks)
ii) Describe two different scenarios in which the generated code fails to compile, and explain what type or error will occur. Give an example for each scenario.

(4 marks)

3. a) Explain the difference between the withincode and cflow pointcuts.

(3 marks)

b) Describe in words what is matched by the following method signatures:
   (i) java.awt..get*(..)
   (ii) * java.awt.Rectangle.getBounds()

(4 marks)

c) Can there be methods that are matched by both signatures given in question 3 b) above? If yes, construct an example, if not, explain why not.

(3 marks)

d) What, if any, is the difference between these four patterns?
   (i) awt.*Widget()
   (ii) awt..*Widget*
   (iii) awt..*Widget
   (iv) awt..*Widget+
   (v) awt.*Widget+

(5 marks)
4. Consider a side-effect free method `fun` that takes an `int` argument and returns an `int`.

a) Write an aspect `LimitFun` that limits the `total` number of calls to `fun` to 1000, and throws an exception `FunLimitExceeded` if this limit is exceeded. Briefly explain your code.

   (3 marks)

b) Since `fun` is side-effect free, it will not terminate if a recursive call uses an argument that has been used before. Implement another aspect `TerminateFun` that detects recursive calls to `fun` with an argument that has been used before, and raises an exception `FunTerminated` if this is the case. Make sure that your code does not accidentally terminate repeated non-recursive calls with the same arguments. Briefly explain your code.

   (9 marks)

c) How would you modify your `FunTerminated` aspect so that it can terminate all methods with one `int` argument and an `int` result?

   (3 marks)
5. a) Consider the six template definitions below:

(I) template <class T1, class T2>
    class P {...};

(II) template <class T1, class T2>
    class P<T1*, T2> {...};

(III) template <class T1, class T2>
    class P<T1, T2*> {...};

(IV) template <class T>
    class P<T, T> {...};

(V) template <class T>
    class P<T*, T*> {...};

(VI) template <>
    class P<int*, int*> {...};

For each of the following five declarations describe which specialization is selected, and why, or describe why the declaration is causing an error.

(i) P<bool, int> a;
(ii) P<int*, int*> b;
(iii) P<bool, int*> c;
(iv) P<int*, bool*> d;
(v) P<int, int> e;

(5 marks)

b) Consider the class MyVector given below. Write a template metaprogram that generates a specialized instance of the mean average of an element of the class MyVector. The mean average of a vector \( x \) of length \( N \) is defined as \( m = \frac{\sum x[i]}{N} \).

```cpp
template<class T, int N>
class MyVector {
    public:
        T operator[](int i) const
        { return data[i]; }

    private:
        T data[N];
};
```

(10 marks)

END OF PAPER