A1(a)

- **Encapsulation** adopts the principle of "information hiding" so that the objects state can only be accessed through the objects operations. The specification (the protocol) is kept separate from the implementation (of the protocol). This encourages better designs that can be reused reliably.
- **Polymorphism** is the application of the same operation to different types of object e.g. the observers in the Observer pattern may respond uniquely to the update() message from the subject. By restricting assumptions that one object makes about another object to a well defined protocol, it facilitates extension (add new classes supporting the protocol) and adaptation (implementation can change within a class). Polymorphism enables like-for-like components to be plugged in and out of a system.
- **Inheritance** is a relationship between classes, which are organised into a hierarchy. Classes lower (sub-classes) in the hierarchy inherit attributes and protocol from classes higher (super-classes) in the hierarchy, then may add additional attributes/operations and/or override operations. Inheritance allows reuse of an existing class, so that only the programming of the differences is required.

A1(b)

Some Object Oriented languages, like Smalltalk demand that every new class is a subclass from an existing class in the overall class hierarchy. This enables the class at the root of the hierarchy to provide a common protocol shared by every class e.g. print state, inspect state, list sub/super-classes. The new subclasses may override the inherited methods. The disadvantage is that every program includes a collection of core classes which makes the binary bigger.

The C^{++} language has been designed so as not to demand the absorption of new classes into the existing class hierarchy which leads to smaller binaries. The downside is that in not being able to inherit a common protocol requires that encapsulation is intentionally broken, in order to allow some classes access to the otherwise private/protected class members. In C^{++} this is done through friend functions. Typically, friend functions are added to a class protocol in order that the I/O stream classes may display state information.

A1(c)

Advantage	Disadvantage
simple to understand and implement, activities are in a linear sequence, each starts after the previous finishes	assumes requirements well-defined, understood and unchanging, have to repeat the whole sequence of steps if requirements change.
widely used and has delivered many successful software projects	feedback (e.g. ambiguity / incompleteness) gained from latter stages of development comes too late since the development cycle is long
easy to track and time project stages	all code written from scratch, does not encourage reuse

Please note that many descriptive answers are longer and more detailed than would be expected in the exam but are included in order to make "teaching points". Use the part-marks as a rough guide as to how much detail is expected.

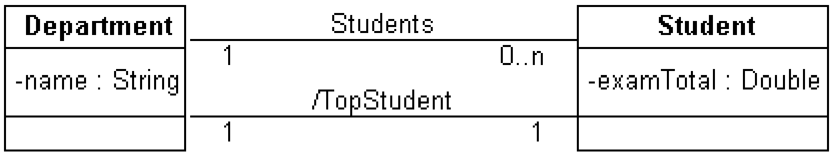
A1(d)

Good dialogue with the domain experts
Classification e.g.Tangible, Role, Event, Organisation
Textual analysis/Grammatical parse - identifying nouns
The Iterative and Incremental process development cycle often reveals new candidate classes

A1(e)

A derived association is an association deduced from other associations, and helps reduce redundant associations which in turn reduces the threat of a difficult to understand or heavily overloaded model. In UML its name is prefixed by a /

e.g. a Department maintains details of all its students thus can generate on demand the derived association /TopStudent



A1(f)

Coupling is a measure of the dependency between classes resulting from collaboration between objects to provide a service. It is desirable to have low coupling between classes, which means that one class is not too dependent on another, which leads to less maintenance and facilitates reuse.

Cohesion is a measure of how strongly related and focussed the responsibilities in a class are. It is desirable that a class exhibit high cohesion, which leads to less maintenance and facilitates reuse.

A1(g)

A) Dataflow e.g. pre-processor -> C^{++} compiler -> C compiler -> assembler -> linker
B) starting code includes: (1) a main program, (2) subclass of TForm available for customisation
C) Palettes = Repository/Database (passive) i.e. DataCentred, editor used for code editor can be swapped in and out i.e. Independent Component, #include <cstdio> to use getchar() i.e. Call and Return, Visual Component Library is layered onto MFC i.e. Virtual

A2(a)

The client module wraps a try-block around the code for normal operation and provides a catch-block to handle each expected exception condition:

```
  try { // normal program code is here...

  }    // error handling code is here ...
    catch (Exception2 o) { etc … }
    catch (Exception1 o) { etc … }
    catch (...)          { etc … }
}
```

The service module *throws* an exception when it detects an error condition. It is then the clients responsibility to decide how to react to this error condition. The exception in an object thus can contain information to help the client to determine how to respond.

```
class MyException{
  public:
    MyException() : name("useful info here"){}

    string getName(void){ return name; }

  private:
     string name;
};

 etc …

MyException o;
throw o; //assume error detected so throw exception
```

A2(b)

```
class Amphibian {
  public:
    virtual void initialise(void)=0;
    int getPosition(void);
    void setPosition(int);
    clColor getColor(void);
    void setColor(clColor);
  protected:
    int position;
    clColor colour;
};


class Frog : public Amphibian {
  public:
    void initialise(void);
};

class HoverFrog : public Frog {
  public:
    int getHeight(void);
    void setHeight(int);
    void initialise(void);
  private:
    int height;
};
```

A2(c)

The TComplex interface is limited to those complex number properties that are needed by the client module. If it became apparent that a larger class was available it is possible that a resourceful programmer of the client module could circumvent the services provided by our TComplex class, so that we could no longer be in control of the consequences. It is quite common to wrap a new class around a class from the standard library in order to limit and control the services provided by the new class

```
template <class A>
class TComplex{
  public:
    TComplex(A r, A i):n(r,i) {}

    complex<A> add(TComplex n1)      { etc … }
    complex<A> minus(TComplex n1)    { etc … }
  private:
    complex<A> n;
};
```

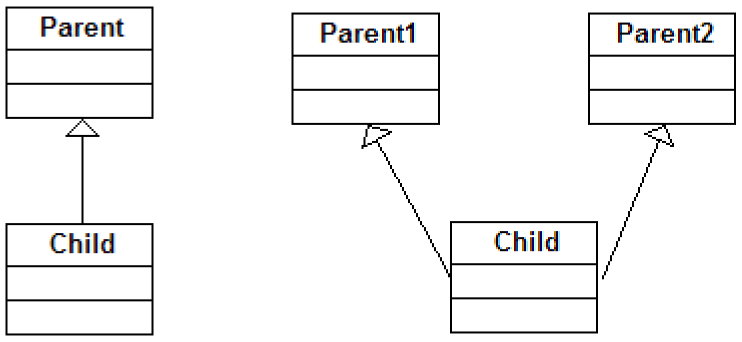
```
TComplex <float>       n1(100.0, 0.0);
TComplex <double>      n2(0.0, 100.0);
TComplex <long double> n3(100.0, 100.0);
```

A2(d)

Multiple (as opposed to single e.g. Parent) inheritance occurs when a sublass has more than one superclass (e.g. Parent1, Parent2).

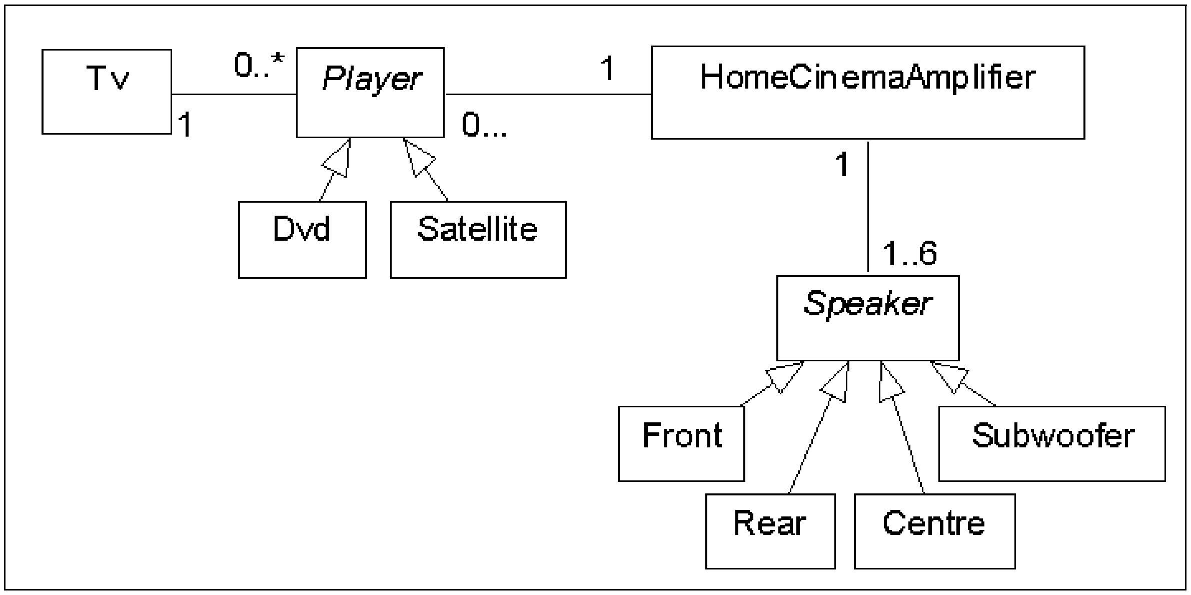
The syntax in C^{++} just requires a list of the superclasses e.g.

```
class Child:
  public Parent1  public Parent2 etc
```



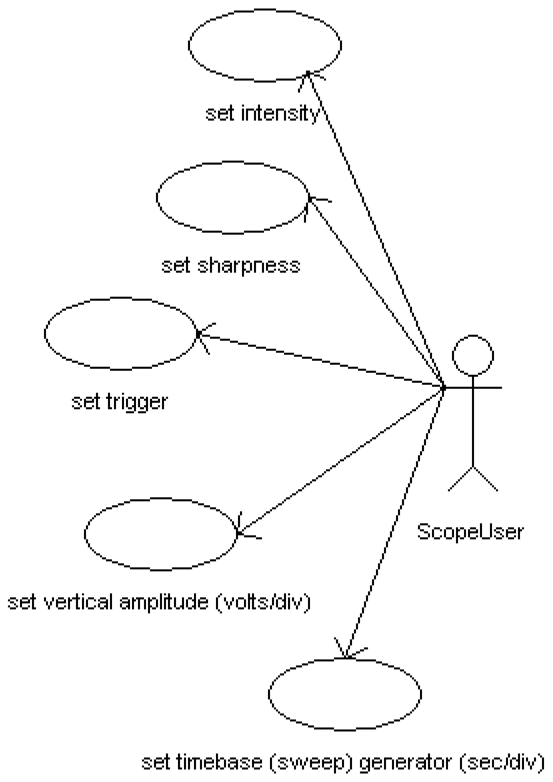
Please note that many descriptive answers are longer and more detailed than would be expected in the exam but are included in order to make "teaching points". Use the part-marks as a rough guide as to how much detail is expected.

A3(a)



A3(b)

Note: system boundary not required since system will not be implemented in software

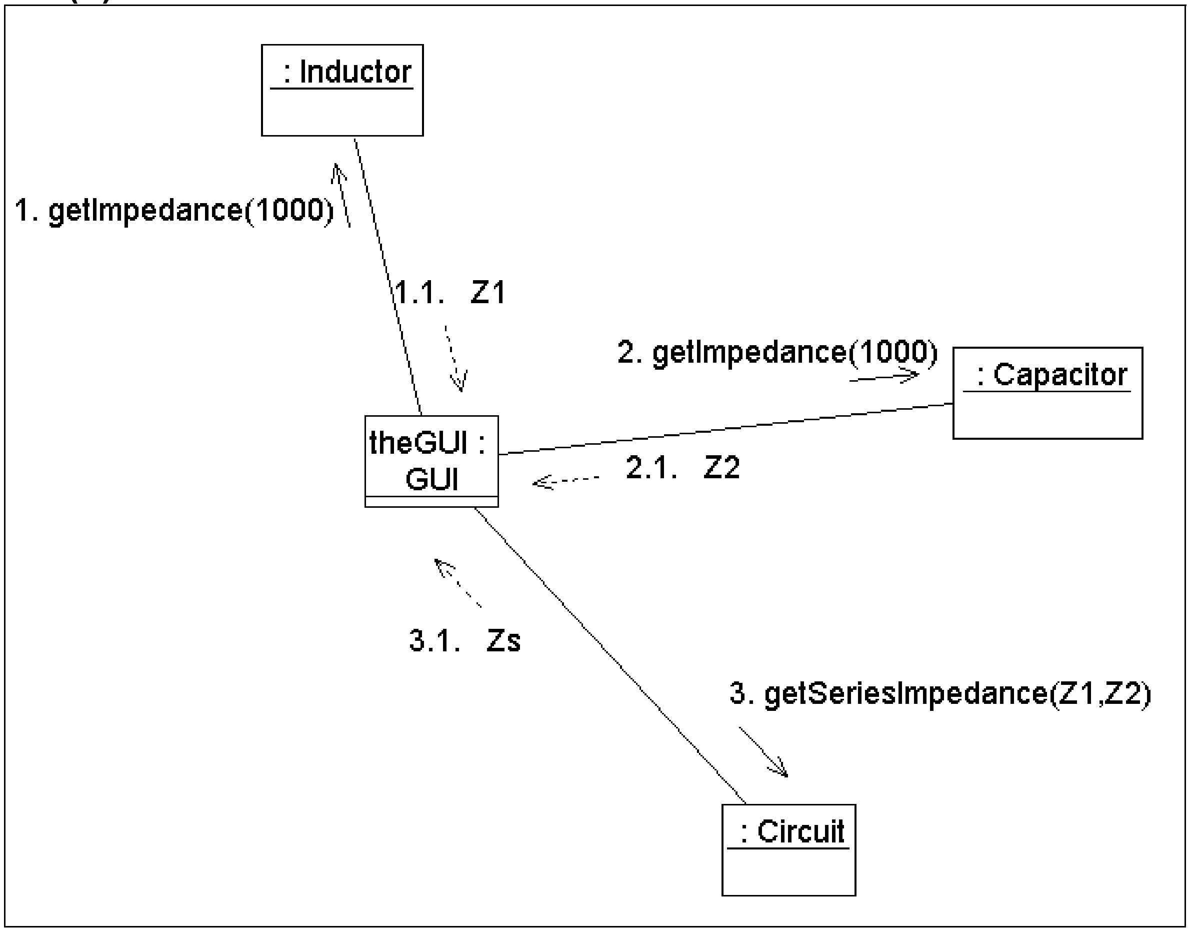

set intensity
set sharpness
set trigger
set vertical amplitude (volts/div)
set timebase (sweep) generator (sec/div)
ScopeUser

Please note that many descriptive answers are longer and more detailed than would be expected in the exam but are included in order to make "teaching points". Use the part-marks as a rough guide as to how much detail is expected.

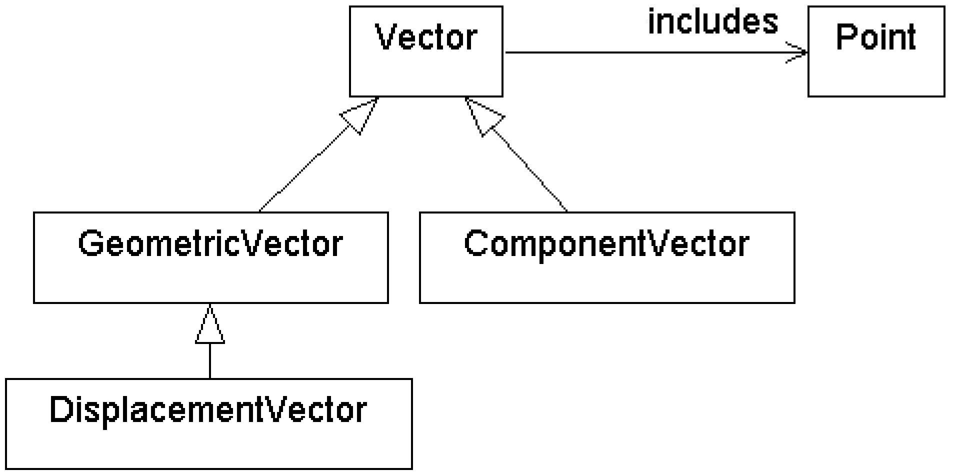
A3(c)

1. omission of the orchestrating instance (which is responsible initiating each use-case i.e. dynamic behaviour)
2. omission of navigation of associations (which can only be determined after consideration of dynamic behaviour)
3. omission of methods (which can only be determined after consideration of dynamic behaviour)

A3(d)



A3(e)



Please note that many descriptive answers are longer and more detailed than would be expected in the exam but are included in order to make "teaching points". Use the part-marks as a rough guide as to how much detail is expected.

A4(a)

A design pattern is a simple and elegant design that captures a general solution to a general problem (i.e. not domain specific) that has been developed over time and is widely applied and accepted as current "best practice".

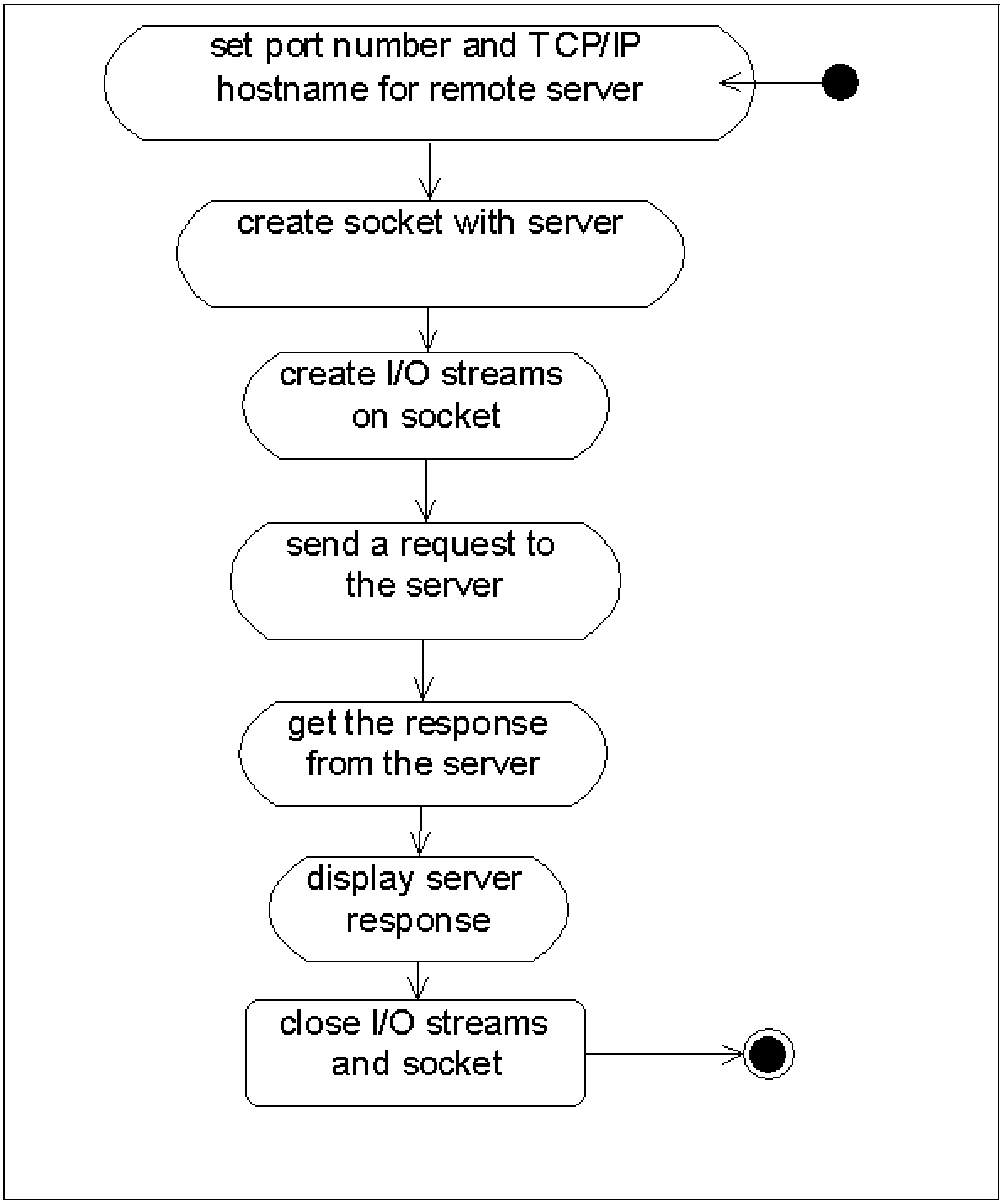
In a small system, the object with the changed state can send messages to its dependents to inform of the state change. However, it is not always logical for an object to *know* details about all its dependents. The Observer pattern is used when there is a one-to-many dependency between objects, such that if there is a state change for a particular object, a number of other objects need be notified. For example, if a file is created in a MS-Console window it is reasonable to expect an MS-Explorer display of the same folder to instantly include the new file. These two applications should be de-coupled so MS-Explorer as are other programs which display folder contents e.g. Compilers, Word-processors would be expected display updated state changes.

The Observer Design Pattern provides a loose coupling between the subject (e.g. folder contents) and the observers (e.g. MS-Explorer). The subject only knows that concrete classes conform to the Observer protocol (including the update() message), but do not know how observers will respond to the messages in that protocol. A drawback is that observers are unaware of each other and can cause message cascades. Not all observers need react to the update() message in the same way, for example a text format change will not affect the word-count in a word-processor. Thus, the receiver of an update() message can respond polymorphically. The Observer design pattern is often used during debugging whilst a program is in development then is de-registered when the program is released.

A4(b)

The client-server architecture reveals the underlying network details e.g. sockets, server-sockets, ports, protocol etc… Thus, it is not a very high-level of abstraction when compared to Object Model architectures e.g. Distributed Objects, Event-based, Tuple-spaces. Performance is good since it is inversely proportional the level of abstraction. However, some performance is lost since the protocol is XML based which is a text format and can be verbose (the Object Model based architectures are all binary. This is compensated by the ease of adaptation of the protocol, since the XML data can include the definition of the protocol.

A4(c)


set port number and TCP/IP hostname for remote server
create socket with server
create I/O streams on socket
send a request to the server
get the response from the server
display server response
close I/O streams and socket

Please note that many descriptive answers are longer and more detailed than would be expected in the exam but are included in order to make "teaching points". Use the part-marks as a rough guide as to how much detail is expected.

A4(d)

Advantages:

- **data acquisition equipment can generate data for processing later**
- **different processing tools can exchange data**
- **data can be embedded in HTML documents and processed by client-side technologies e.g. Javascript**
- **etc…**

```
<!-- XML data for two signals -
     Note: only one requested in the question  -->

<SIGNALLIST>

    <SIGNAL>
        <Description>
              Sampled signal: x[n]=1+sin(nPI/4)+2cos(nPI/2)
        </Description>

        <Domain>Time</Domain>

        <NumberDataValues>8</NumberDataValues>

        <DataValues Complex="NO">3.0 etc… </DataValues>
    </SIGNAL>

    <SIGNAL>
        <Description>
           Complex spectral harmonics x[k]
           obtained from DFT of x[n]
        </Description>

        <Domain>Frequency</Domain>

        <NumberDataValues>8</NumberDataValues>

        <DataValues Complex="YES">1.0 etc… </DataValues>
    </SIGNAL>

</SIGNALLIST>
```

Please note that many descriptive answers are longer and more detailed than would be expected in the exam but are included in order to make "teaching points". Use the part-marks as a rough guide as to how much detail is expected.