IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2010

ISE PART II: MEng, BEng and ACGI

SOFTWARE ENGINEERING 2

Tuesday, 1 June 2:00 pm

Time allowed: 2:00 hours

There are FOUR questions on this paper.

Q1 is compulsory.
Answer Q1 and any two of questions 2-4.
Q1 carries 40% of the marks. Questions 2 to 4 carry equal marks (30% each).

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible First Marker(s) : L.G. Madden, L.G. Madden
Second Marker(s) : J.V. Pitt, J.V. Pitt

The Questions

1. **[Compulsory]**

All the parts of this question refer to Figure 1.1 which is based on the Analogue filter case study developed during the E2.12 laboratories i.e. it is similar but not identical to the code used for laboratories. Your answers should be based only on what is shown in this question.

a) Your UML diagrams should be based only on what is shown in the figure.

i) Draw a class diagram showing the class architecture.

[10]

ii) Draw an interaction diagram showing the main program behaviour including instantiations. Only show the first iteration of the `for` loop. You do not need to calculate any values, invent typical values where needed.

[6]

iii) Draw an object diagram showing the state of the polymorphic variable in the main program after it has been instantiated.

[4]

b) Use fully qualified names for your answers where appropriate. When you are asked to identify something that you believe is *not* present in the code then you should state "*None*" for your answer.

i) Identify *three different* kinds of relationships between classes. For each example name two classes involved in the relationship.

[3]

ii) Briefly explain why `TBandPassFilter` demonstrates *three different* mechanisms for the provision of *application* member function code in a *derived* class.

[3]

iii) Identify the OCCF members in any one of the classes.

[4]

c) For this question part you should write C++ code for an *implementation* file. In both answers you should *invoke* base class code. Assume that all the implementation code for the base class has been written.

i) Provide implementation code for the constructor `TBandPassFilter(double,double,double)`

[2]

ii) Provide implementation code for the member function `TBandPassFilter::getQ()`. Note: the calculation uses the inverse of the equation for the base class filter.

[2]

d) Rewrite the main program extract using an iterator (i.e. instead of a loop index variable) and an aggregate class (i.e. instead of an array).

[6]

```
class TFilter {        // extract from class definitions follows …
protected:
    double rVal, lVal, cVal;
    TFilter(void)                                                    // etc…
    TFilter(const TFilter &tf)                                       // etc…
    ~TFilter(void)                                                   // etc…
    TFilter & operator=(const TFilter &f)                            // etc…
    TFilter(double rVal, double lVal, double cVal)                   // etc…
public:
    double getF0(void)                                               // etc…
    virtual double getQ(void)                                        // etc…
    virtual complex <double> getGain(double) = 0;                    // etc…
}; // end TFilter

class TCircuit {
private:
    complex<double> comp1, comp2;                                    // etc…
}; // end TCircuit

class TBandPassFilter : public TFilter {
public:
    TBandPassFilter(double rVal, double lVal, double cVal)           // etc…
    double getQ(void)                                                // etc…
    complex<double> getGain(double freq)                             // etc…
}; // end TBandPassFilter

int main(void) {            // extract from main program follows …
    typedef TFilter * TFilterType[];
    TFilterType filters = {new TBandPassFilter(1.0,2.0,1.0),
                           new TBandPassFilter(2.0,1.0,2.0)};
    complex <double> z1(0.0,0.0);
    for (int i = 0; i < 2; i++) {
        (z1 = filters[i]->getGain(1.0)).real();                      // etc…
```

Figure 1.1 Code extract from an application involving analogue filters.

2. The aim of this question is to explore the concept of C++ operator overloading. All the parts of this question refer to Figures 2.1 and 2.2 which are based on the Analogue filter case study developed during the E2.12 laboratories i.e. they are similar but not identical to the figures used for laboratories. Your answers should be based only on what is shown in this question.

a)

i) Briefly describe the difference between overloading and overriding.

ii) Excluding overloaded operators, what other types of function can be overloaded in C++?

iii) When comparing an assignment operator with a copy constructor identify one feature that is *similar* and one feature that is *not similar.*

iv) Identify one operator that *must be* implemented as a member function.

v) Identify one operator that *can not* be implemented as a member function.

vi) How do we decide that an operator *can not* be implemented as a member function for a user-written class?

[9]

b) Figure 2.1 shows a partial specification for the `TComplex` class. For each part (i)-(iii) you should specify any changes to the protocol of the class `TComplex` that is required in order to complete the coding of the overloaded operator. If no changes are required then specify "No changes required".

i) Write C++ implementation code for a *stream insertion* operator for the `TComplex` class as a *friend function.*

ii) Write C++ implementation code for a *stream extraction* operator for the `TComplex` class as an *ordinary function.*

iii) Write C++ implementation code for an *addition* operator for the `TComplex` class as a *member function.*

[15]

c) Provide a single assertion statement that unit tests the addition function of the `TComplex` class. Use the `assertEqualsState()` function shown in Figure 2.2 (Note: choose simple values for the complex addition).

i) Using the `addComplex()` member function.

ii) Using the overloaded addition operator.

iii) Identify a design principle demonstrated by Figure 2.1. Briefly, justify your answer.

iv) Identify a design principle demonstrated by Figure 2.2. Briefly, justify your answer.

[6]

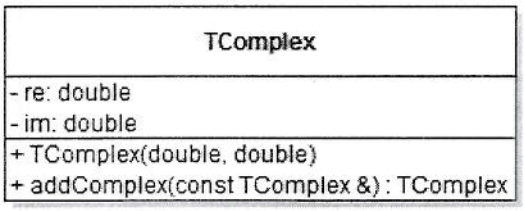


Figure 2.1 Partial specification for `TComplex`

```
bool assertEqualsState(string oKmessage,
                       TComplex expectedValue,
                       TComplex actualComputedValue) {
/*
 * pre    true
 * post   returns true if the state of the two complex numbers are the same
 *        i.e. corresponding real and imaginary cofficients are
 *        the same. Otherwise, returns false.
 */
```

Figure 2.2 Ordinary function specification for `TComplex` state comparison

3. All the parts of this question refer to Figure 3.1 which is based on the problem domain description for the robot world case study developed during the E2.12 tutorials i.e. it is similar but not identical to the description used for tutorials. Your answers should be based only on what is shown in this question.

a) Based on a textual analysis of the problem description suggest names for four candidate classes. Briefly describe the role of each of your four suggestions.

[4]

b) Briefly justify the need for a fifth class that is not explicit in the problem description but is implicit because it can be used to factor out shared structure and behaviour. Briefly explain why this class should be included as an association OR as a generalisation relationship.

[3]

c) Using your answers to (a) and (b) draw a UML class diagram that could be passed on to a programmer for implementation in code. Include as much detail as can be found in the problem description and no more.

[13]

d) Using your answer to (c) give an example of a one-to-one association and an example of a one-to-many association. In both cases you can either name the classes explicitly and/or annotate your class diagram to indicate the two examples.

[2]

e) Give an example of a requirement in the problem description that can not be recorded visually in the class diagram. Briefly describe how this requirement can be recorded.

[2]

For parts (f) – (h) when you are asked to identify something that you believe is *not* present in the class diagram then you should state "*None*" for your answer.

f) Identify an association in your answer to (c) could be made into a derived association (if any). Briefly justify your answer.

[2]

g) Identify an association in your answer to (c) could be made into an association class (if any). Briefly justify your answer.

[2]

h) Identify an association in your class diagram (if any) could be made into a qualified association. Briefly justify your answer.

[2]

A robot moves around a two-dimensional grid. The position of a robot on the grid is expressed as x and y coordinates (integer values). A robot records its current position on the grid. A robot can respond to a number of messages. For example, a robot object may be asked to:

- move one unit over the grid in a north or south or east or west direction;
- indicate its current x position;
- indicate its current y position

A robot may be free standing on the grid or it may be loaded onto a robot carrier for transportation to a grid location in order to perform some task e.g. landmine detection. The robot carrier may carry up to four robots at a time. Like a robot, the robot carrier records its current position on the grid. Robots loaded onto the carrier have the same location as the carrier.

A robot carrier can respond to the same movement messages and location reporting messages as that of an individual robot. A robot carrier can also respond to a number of additional messages. For example, a robot carrier object may be asked to:

- indicate how many places are available for carrying robots i.e. 0 to 4;
- add a robot to the carrier;
- remove a robot from the carrier;
- move to an arbitrary x, y position on the grid.

Every robot has a finite amount of semi-conductor memory. The quantity of installed memory when the robot is constructed determines the type of the robot. A team leader robot has a large amount of memory and a drone robot has a small amount of memory. There must be exactly one team leader and one to three drone robots on the robot carrier before it moves to a new location.

A drone robot can also respond to a number of additional messages. For example, a drone robot object may be asked to:

- Harvest data (associated with the task to be performed by the drone);
- Transfer the data to the team leader robot.

A team leader robot can also respond an additional message. For example, a team leader robot object may be asked to:

- Receive and store all the data harvested by the drone robots;

Figure 3.1 Problem domain description for the robot world case study

4. All the parts of this question refer to Figures 4.1 and 4.2 which are based on the nodal analysis study developed during the E2.12 tutorials i.e. they are similar but are not identical to the figures used for tutorials. Your answers should be based only on what is shown in this question.

Figure 4.1 shows a simple resistive circuit, the associated analysis and a solution coded in Matlab™. Figure 4.2 shows a partial specification for the class `TSequence<XYZ>`.

a) Translate the UML class diagram shown in Figure 4.2 into C++ class *specification code* using a template type `<XYZ>`.

[6]

b) Using the protocol shown in Figure 4.2 draw an interaction diagram that implements the solution demonstrated by the Matlab™ code shown in Figure 4.1. The code calculates the unknown currents. Your interaction diagram should show all instantiations. Use the same identifiers and values as shown in the Matlab™ code.

[6]

c) A robust C++ program should check that the resistors matrix is not singular before attempting to invert it. Assuming that all standard operators have been overloaded, provide a short C++ code extract from a *client* application that demonstrates the check on the matrix and the matrix inversion using the following approaches:

i) Using an assertion.

ii) Using design by contract.

iii) Using defensive programming.

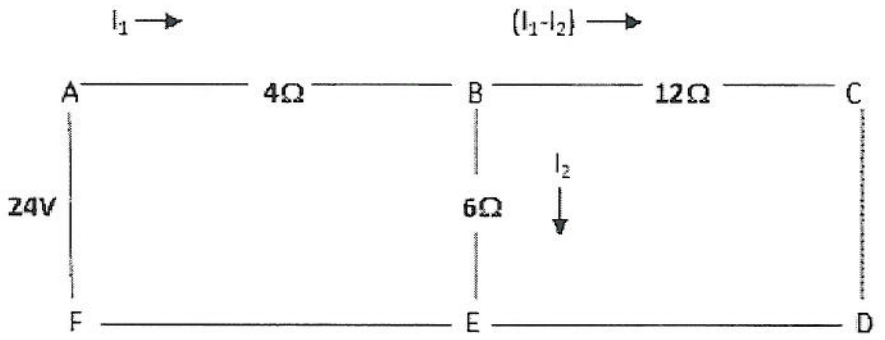
[6]

d) For this question part you should write C++ code for a specification file for each of the two new architectures described below. You do not need to include OCCF or any other members not originally shown in Figure 4.2 but you may need to modify some of the signatures for the new classes.

i) Replace the original `TSequence<XYZ>` class with its members distributed between two template classes based on matrix algebra. That is a 2 by 2 matrix and a 2 by 1 vector. There is an association relationship between the two classes.

ii) Refactor the original `TSequence<XYZ>` class so that it is now a base class to the two matrix algebra classes created for (i) i.e. they are now derived classes. There is a generalisation relationship between the three classes.

[12]



Using Kirchoff's Laws for the loop ACDF: $16* I_1 - 12* I_2 = 24$ (Eqn. 4.1)

and for the loop ABEF: $4* I_1 + 6* I_2 = 24$ (Eqn. 4.2)

Solving (Eqn. 4.1) and (Eqn. 4.2) gives $I_1 = 3$ and $I_2 = 2$

The simulataneous equations may be solved in Matlab™ as follows

```
>> resistorsMatrix = [16 -12; 4 6];
   voltagesVector = [24; 24];
   resistorsInverted = resistorsMatrix ^-1;
   currentsVector = resistorsInverted * voltagesVector
```

Note: ^-1 represents 2x2 matrix inversion.

Figure 4.1 Nodal analysis of a simple resistive circuit

TSequence <XYZ>
noRows : int # noCols : int # theData[ ] : XYZ
+ TSequence <XYZ> (int rows, int cols) + put (int row, int col, XYZ value) : void + at (int row, int col) : XYZ + determinant(void) : XYZ + notSingular(void) : bool + multiplyMatrixByVector(const TSequence <XYZ> & matrix2by2) + invert(void) : TSequence <XYZ>

Note: The class `TSequence` may be used to represent a 2 by 1 vector or a 2 by 2 matrix but the elements of the vector/matrix are stored in a linear sequence.

Note: the formal argument for `TSequence::multiplyMatrixByVector()` is a 2 by 2 matrix and the result is a 2 by 1 vector.

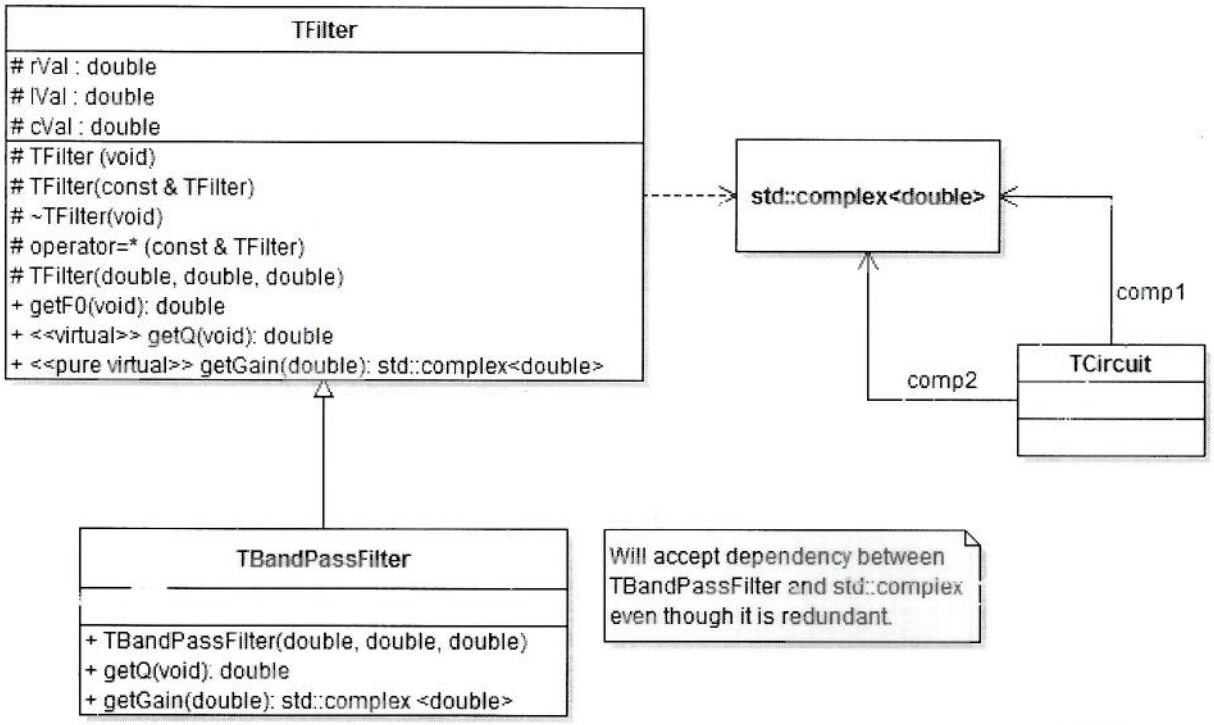
Note: `TSequence::notSingular()` returns `true` if a matrix is not singular, otherwise, returns `false`.

Figure 4.2 Partial class specification for `TSequence<XYZ>`

1/14

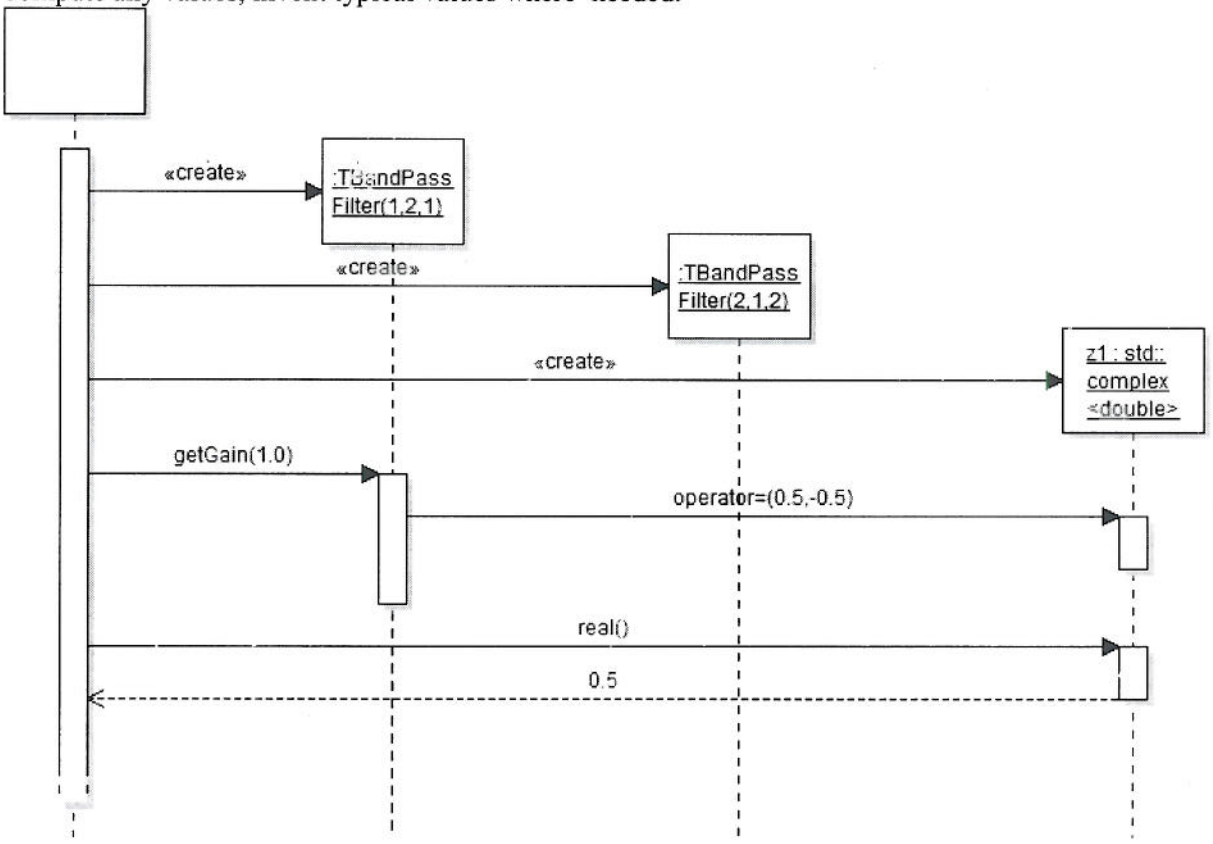
a) Your UML diagrams should be based only on what is shown in the figure.

i) Draw a class diagram showing the class architecture.



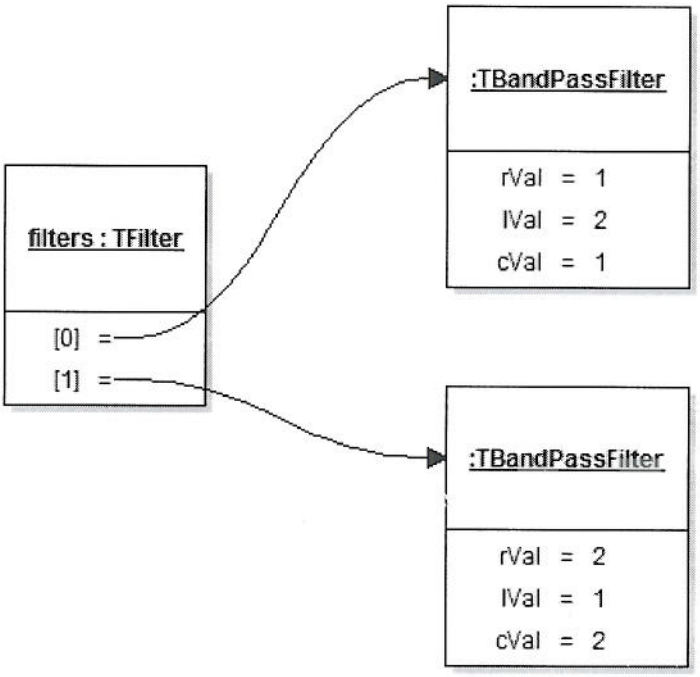
[10]

ii) Draw an interaction diagram showing the main program behaviour including instantiations. Only show the first iteration of the `for` loop. You do not need to compute any values, invent typical values where needed.



[6]

iii) Draw an object diagram showing the state of the polymorphic variable in the main program after it has been instantiated.



[4]

b) Use fully qualified names for your answers where appropriate. When you are asked to identify something that you believe is *not* present in the code then you should state "*None*" for your answer.

i) Identify *three different* kinds of relationships between classes. For each example name two classes involved in the relationship.

1. Inheritance between `TFilter` and `TBandPassFilter`
2. Association between `TCircuit` and `std::complex<T>`
3. Dependency between `TFilter` (or `TBandPassFilter`) and `std::complex<T>`

[3]

ii) Briefly explain why `TBandPassFilter` demonstrates *three different* mechanisms for the provision of *application* member function code in a *derived* class.

1. `TBandPassFilter::getF0()` demonstrates drop-through inheritance from `TFilter::getF0()`
2. `TBandPassFilter::getQ()` demonstrates overridden inherited implementation code from `TFilter::getQ()`
3. `TBandPassFilter::getGain()` demonstrates overridden inherited specification from `TFilter::getGain()`

[3]

iii) Identify the OCCF members in any on of the classes.

[4]

The OCCF members are:

```
TFilter::TFilter(void)
TFilter::TFilter(const & TFilter)
TFilter::~TFilter(void)
TFilter::operator=(const & TFilter)
```

c) For question part (c) you should write C++ code for an *implementation* file. In both answers you should *invoke* base class code. Assume that all the implementation code for the base class has been written.

i) Provide implementation code for the constructor `TBandPassFilter(double,double,double)`

```
TBandPassFilter::TBandPassFilter(
        double rVal,
        double lVal,
        double cVal) : TFilter(rVal, lVal, cVal) {}
```

[2]

ii) Provide implementation code for the member function `TBandPassFilter::getQ()`. Note: the calculation uses the inverse of the equation for the base class filter.

```
double TBandPassFilter::getQ(void) {
    return 1.0 / TFilter::getQ();
}
```

[2]

d) Rewrite the main program extract using an iterator (i.e. instead of a loop index variable) and an aggregate class (i.e. instead of an array).

```
typedef TFilter * TFilterType[];
TFilterType filters = {new TBandPassFilter(1.0,2.0,1.0),
                       new TBandPassFilter(2.0,1.0,2.0)};
complex <double> z1(0.0,0.0);
```

```
typedef vector <TFilter *> TFilterType;
TFilterType filters;
filters.push_back(new TBandPassFilter(1.0, 2.0, 1.0));
filters.push_back(new TBandPassFilter(2.0, 1.0, 2.0));
complex <double> z1(0.0,0.0);
```

```
for (int i = 0; i < 2; i++) {
TFilterType::iterator index;
for (index=filters2.begin(); index!=filters2.end();++index)

   (z1 = filters[i]->getGain(1.0)).real();
   (z1 = (*index)->getGain(1.0)).real();
```

[6]

2. The aim of this question is to explore the concept of C^{++} operator overloading. All the parts of this question refer to Figure 2.1 which is similar, but is not identical to the Analogue filter case study developed during the E2.12 Computing laboratories.

a)

i) Briefly describe the difference between overloading and overriding?

Overloading involves the same function name but with different signatures.

Overloading involves the same function name but with the same signature which occurs in an inheritance relationship

ii) Excluding overloaded operators, what other types of function can be overloaded in C^{++}?

Constructors and ordinary functions

iii) When comparing an assignment operator with a copy constructor identify one feature that is *similar* and one feature that is *not similar*.

Both clone state. Copy constructor is associated with object instantiation but assignment is associated with any part of an objects lifecycle.

iv) Identify one operator that *must be* implemented as a member function.

assignment operator

v) Identify one operator that *can not* be implemented as a member function?

stream insertion or extraction operator

vi) How do we decide that an operator *can not* be implemented as a member function in a user-written class?

if the first operator argument is not an instance of the user-written class.

[9]

b) For each part (i)-(iii) you should specify any changes to the protocol of the class `TComplex` that is required in order to complete the coding. If no changes are required then specify "No changes required".

i) Write C++ implementation code for a *stream insertion* operator for the `TComplex` class as a *friend function*.

```
ostream& operator<< (ostream &o, const TComplex &tc){
   o << tc.re  << "+" << tc.im << "j";
   return o;
}
```

Add the following member to the class definition:

```
friend ostream& operator<<(ostream&, const TComplex &);
```

ii) Write C++ implementation code for a *stream extraction* operator for the `TComplex` class as an *ordinary function*.

```
istream& operator>> (istream &i, TComplex &v){
   double aValue;
   i >> aValue; v.setReal(aValue);
   i >> aValue; v.setImag(aValue);
   return i;
}
```

Add the following members to the class definition:

```
void setReal(int aValue);
void setImag(int aValue);
```

iii) Write C++ implementation code for an *addition* operator for the `TComplex` class as a *member function*.

```
TComplex TComplex::operator+ (const TComplex &tc){
  this->re= this->re+tc.re; this->im= this->im+tc.im;
// or this->addComplex(tc)
  return *this;
}
```

Add the following member to the class definition:

```
TComplex & operator+ (const TComplex &);
```

[15]

c) Provide a single assertion statement that unit tests the addition function of the `TComplex` class. Use the `assertEqualsState()` function shown in Figure 2.2 (Note: choose simple values for the complex addition).

i) Using the `addComplex()` member function.

```
assert(assertEqualsState("Complex MF add test  - OK",
      TComplex(4.0, 6.0),
      TComplex(1.0, 2.0).addComplex(TComplex(3.0, 4.0))
));
```

ii) Using the overloaded addition operator.

```
assert(assertEqualsState("Complex OO add test  - OK",
      TComplex(4.0, 6.0),
      TComplex(1.0, 2.0) + TComplex(3.0, 4.0)
));
```

[6]

3.

a) Based on a textual analysis of the problem description suggest names for four candidate classes. Briefly describe the role of your four suggestions.

`TRobot, TRobotDrone, TRobotLeader, TRobotCarrier.`

[4]

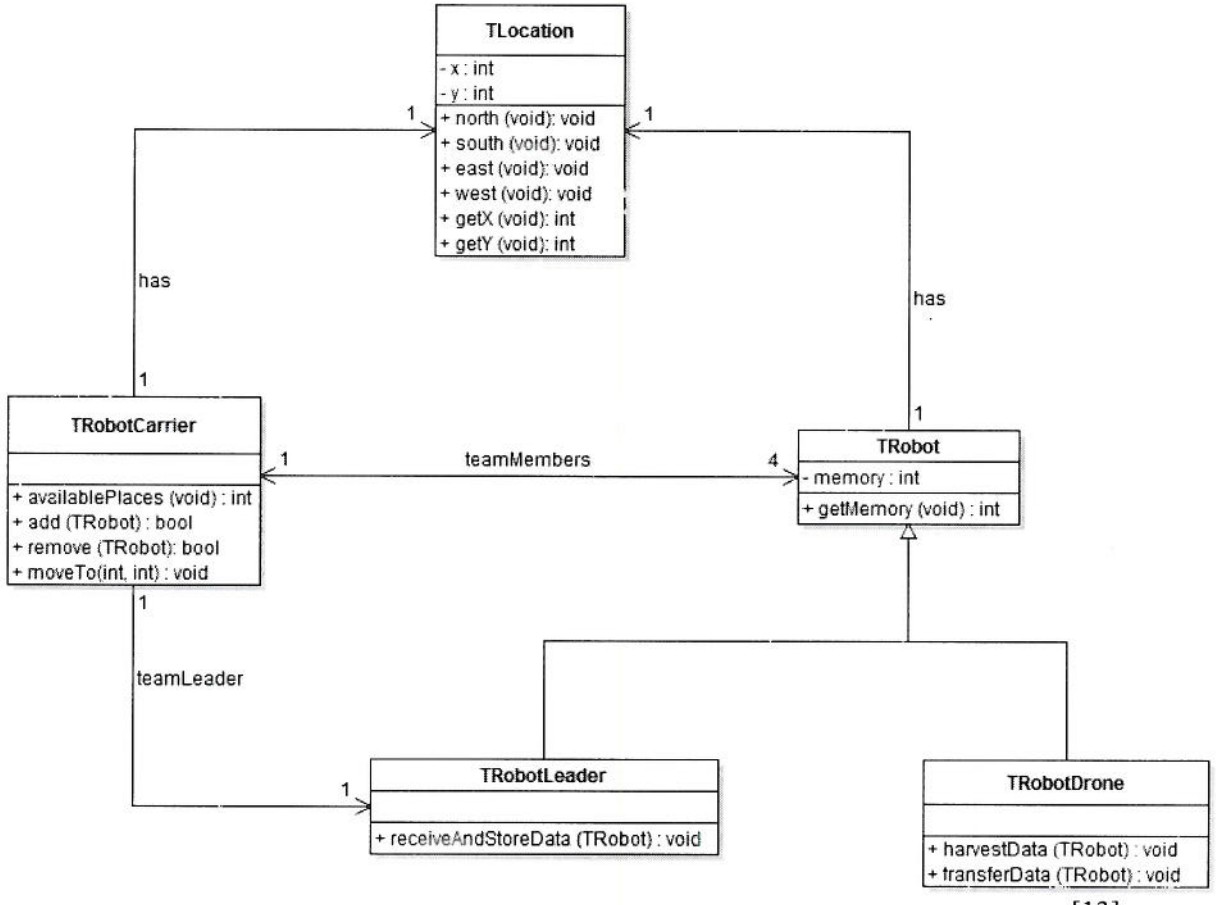
b) Briefly justify the need for a fifth class that is not explicit in the problem description but is implicit because it can be used to factor out shared structure and behaviour. Briefly explain why this class should be included as an association OR as a generalisation relationship.

`TLocation` can be used to represent grid position and the basic movement messages. Justification is either

- Both `TRobot` and `TRobotCarrier` *has-a* location suggesting an association.
- There is no logical relationship between `TLocation` and `TRobot`/ `TRobotCarrier` which would suggest an inheritance (is-a) relationship.

[3]

c) Using your answers above draw a UML class diagram that could be passed on to a programmer for implementation in code. Include as much detail as can be found in the problem description and no more.



[13]

d) Using your diagram give an example of a one-to-one association and an example of a one-to-many association. In both cases you can either name the classes explicitly or clearly annotate your class diagram to indicate the two examples.

One-to-one: `TRobotCarrier` and `TRobotLeader`

One-to-many: `TRobotCarrier` and `TRobot`

[2]

e) Give an example of a detail in the problem description that can not be recorded visually in the class diagram. Briefly describe how this detail is recorded.

"There must be exactly one team leader on a robot carrier before it moves to a new location". Visually, we can show that a robot carrier carries up to four robots and that a robot carrier carries a team leader. But we can not show that one of the robots must be a team leader. To record this detail we need to augment the diagram with a textual description of an invariant.

[2]

f) Which association in your class diagram (if any) could be made into a derived association? Briefly justify your answer.

The `teamleader` association could be derived. We could write a routine to iterate through the robots on the carrier looking for the one with a lot of memory.

[2]

g) Which association in your class diagram (if any) could be made into an association class? Briefly justify your answer.

[2]

The `teammembers` association could be promoted to an association class if we wanted to record extra details e.g. a team name, work activity etc....

h) Which association in your class diagram (if any) could be made into a qualified association? Briefly justify your answer.

The `teammembers` association could be qualified association if each robot was given a unique indentifier.

[2]

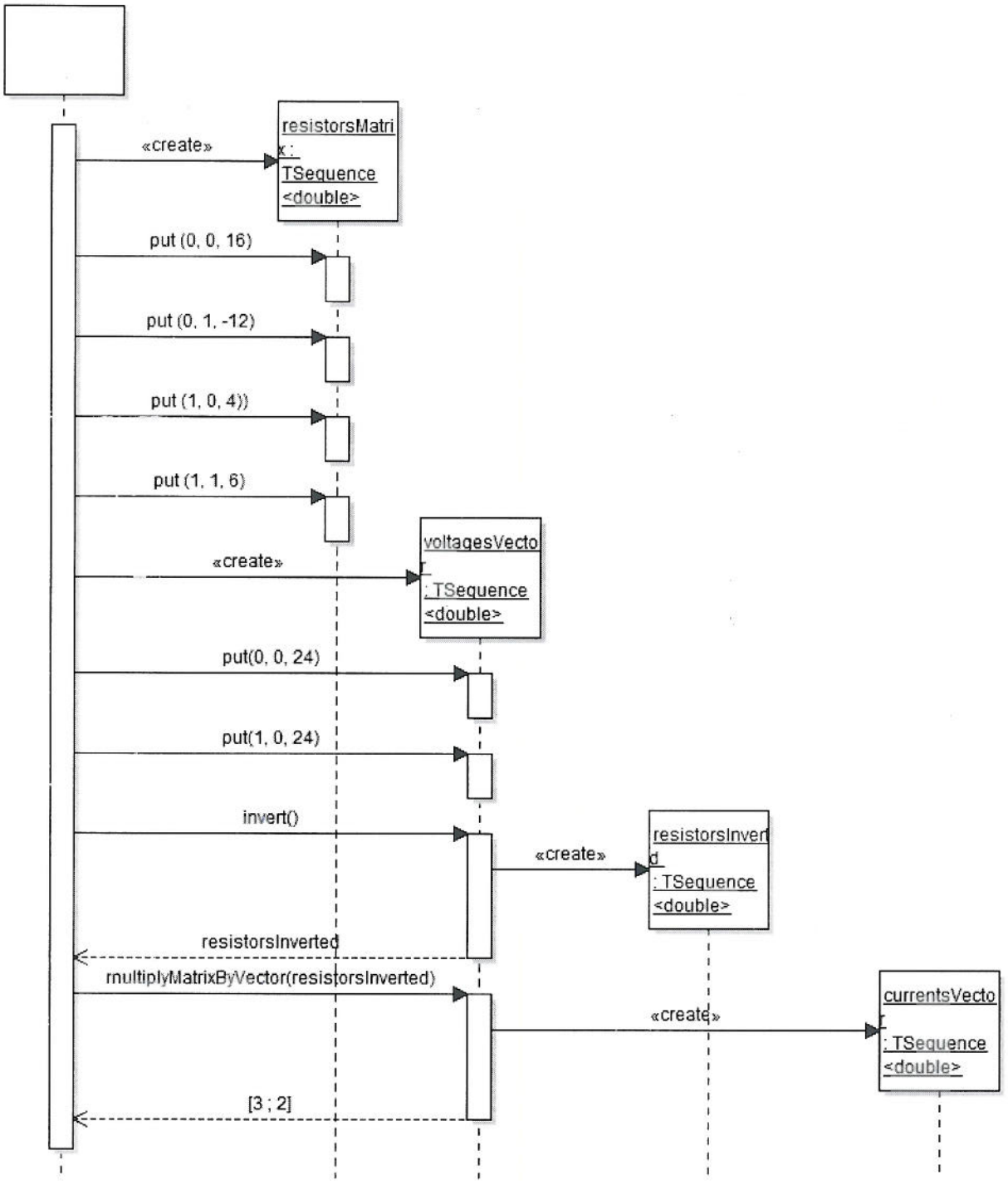
4.

a) Translate the UML class diagram shown in Figure 4.2 into C++ class *specification code* using a template type.

[6]

```
template <typename XYZ> class TSequence
public:
    TSequence<XYZ> (int rows,int cols);
    void put(int row,int col, XYZ value);
    XYZ at(int row, int col);
    XYZ determinant(void);
    bool notSingular(void);
    TSequence<XYZ> invert(void);
    TSequence<XYZ> multiplyMatrixByVector(
                                    TSequence<XYZ> matrix2by2);
protected:
    XYZ *theData; //Used as a 1 or 2-dimensional array
    int noRows; //No.rows initialised by constructor 1st arg
    int noCols; //No.cols initialised by constructor 2nd arg
};
```

b) Using the protocol shown in Figure 4.2 draw an interaction diagram that implements the solution demonstrated by the Matlab™ code shown in Figure 4.1. The code calculates the unknown currents and displays the results. Your interaction diagram should show all instantiations. Use the same identifiers as used in the Matlab™ code. [6]



c) A robust C++ program should check that the resistors matrix is not singular before attempting to invert it. Assuming that all standard operators have been overloaded, provide a short C++ code extract from a *client* application that demonstrates the check on the matrix and the matrix inversion in the following approaches:

i) Using an assertion.

```
assert(resistorsMatrix.notSingular() == true)
resistorsInverted = resistors.invert();
```

ii) Using design by contract.

```
if (resistorsMatrix.notSingular() == true)
    resistorsInverted = resistors.invert();
```

iii) Using defensive programming.

```
resistorsInverted = resistors.invert();
```

[3]

d) For question part (d) you should write C++ code for a *specification* file for each of the two new architectures described below. You do not need to include OCCF or any other members not originally shown in Figure 4.2 but you may need to modify some of the signatures for the new classes.

i) Replace the original `TSequence` class with its members distributed between two classes based on matrix algebra. There is an association relationship between the two classes.

```
template <typename XYZ> class TMatrix;
// not expected but forward declaration needed for compilation

template <typename XYZ> class TVector{
public:
 TVector <XYZ> (void);
 void put (int col, XYZ value);
 XYZ at(int col);
 TVector<XYZ> multiplyMatrixByVector(TMatrix<XYZ> matrix2by2);
private:
 XYZ theData[2];
};

template <typename XYZ> class TMatrix{
public:
  TMatrix <XYZ> (void);
  void put (int row, int col, XYZ value);
  XYZ at(int row, int col);
  XYZ determinant(void);
  bool notSingular(void);
  TMatrix <XYZ> invert(void);
private:
  TVector<XYZ> rows[2];
};
```

ii) Refactor the original `TSequence` class so that it is now a base class to the two matrix algebra classes created for (i) i.e. they are now derived classes. There is a generalisation relationship between the three classes.

```
template <typename XYZ> class TSequence{
public:
    TSequence<XYZ> (int rows,int cols);
    void put(int row,int col, XYZ value);
    XYZ at(int row, int col);
protected:
    XYZ *theData; //Used as a 1 or 2-dimensional array
    int noRows; //No.rows initialised by constructor 1st arg
    int noCols; //No.cols initialised by constructor 2nd arg
};

template <typename XYZ> class TMatrix;
// not expected but forward declaration needed for compilation

template <typename XYZ> class TVector : public TSequence <XYZ>
      {
public:
    TVector <XYZ> (void);
    TVector<XYZ> multiplyMatrixByVector(TMatrix<XYZ>
      matrix2by2);
};

template <typename XYZ> class TMatrix : public TSequence  <XYZ>
      {
public:
    TMatrix <XYZ> (void);
    XYZ determinant(void);
    bool notSingular(void);
    TMatrix <XYZ> invert(void);
};
```