IMPERIAL COLLEGE LONDON

Final Copy.

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2007

ISE PART II: MEng, BEng and ACGI

SOFTWARE ENGINEERING 2

Tuesday, 29 May 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

1. This question is based on Figure 1 which is a syntactically correct C++ program that demonstrates a number of Object Oriented concepts, C++ features and Design Principles. Note that any answers to the following question that require UML diagrams may use UML textual stereotypes to add clarity or detail.

(a) Examine all the code and name a single example (if any) for each of the Object Oriented Concepts listed below (expressed in C++). If you think there is no example, state "None". If there is an example then specify an identifier (using the Scope Resolution Operator *if necessary*) e.g. `istream::operator>>`, `AB::doit()`.You do not need to explain the meaning of the terms in (a) and (b).

(i) a polymorphic variable,
(ii) a polymorphic member function,
(iii) a constructor or member function that has been overridden,
(iv) a constructor or member function that has been overloaded,
(v) a class constant,
(vi) an abstract class. [6]

(b) Examine all the code and name a single example (if any) for each of the C++ features listed below. Use the same notation for you answer as in (d).

(i) a first class container class in the standard library,
(ii) an overloaded operator for a standard library class,
(iii) a standard library class that has a template data member,
(iv) a member function that is found in all first class container classes,
(v) a non-mutating generic algorithm,
(vi) a pseudo-variable. [6]

(c) Examine all the code and identify one example of the application of each of the Design Principles below. Briefly describe why your example is a demonstration of the Design Principle. Reference the relevant line numbers in your answer.

(i) Least Privilege,
(ii) Information Hiding and
(iii) Separation of Concerns. [9]

(d) Predict the output of the program as it would appear on the console. [1]

(e) Examine all the code and draw the corresponding UML class diagram. Use the standard symbols for the visibility modifiers i.e. + # and - [6]

(f) Examine the `main()` function and draw the corresponding UML Object Diagram showing the final state of the object `a1` in the function. [4]

(g) Examine the `main()` function and draw the corresponding UML Sequence Interaction Diagram for the body of the function. [8]

```
class A {                                                          // 1
   public:                                                         // 2
      A(void):ch(NULL){};                                          // 3

      A(const char ch):ch(ch){}                                    // 4

      virtual const bool doit(void)const = 0;                      // 5

   protected:                                                      // 6
      const char ch;                                               // 7
};                                                                 // 8

class AB : public A{                                               // 9
   public:                                                         //10
      AB(const char ch):A(ch){}                                    //11

      const bool doit(void) const {                                //12
        return (DIGITS.find(this->ch) != string::npos);            //13
      }                                                            //14

   private:                                                        //15
      const static string DIGITS;                                  //16
  };                                                               //17

  const string AB::DIGITS = "0123456789";                          //18

class AC : public A{                                               //19
   public:                                                         //20
      AC(const char ch):A(ch){}                                    //21

      const bool doit(void) const{                                 //22
        return ((this->ch >= '0') && (this->ch <= '9'));           //23
      }                                                            //24
};                                                                 //25

int main(void){                                                    //26
   typedef vector <A *> TV;                                        //27
   TV a1;                                                          //28
   TV::iterator i;                                                 //29
   AB ab1('8');                                                    //30
   AC ac1('?');                                                    //31
   a1.push_back(&ab1);                                             //32
   a1.push_back(&ac1);                                             //33
   for (i=a1.begin(); i != a1.end(); i++)                          //34
      cout << boolalpha << (*i)->doit() << endl;                   //35
}
```

Figure 1

2. The aim of this question is to identify and replace residual C code in order to benefit from the reusable and adaptable features provided by C++. Some of these features are unavailable in C and some duplicate functionality available in C. Figure 2.1 demonstrates a program that is written in a subset of C++ and will compile successfully in a C or a C++ compiler. Figure 2.2 shows a screen dump for a typical interaction with the program.

The commented lines in Figure 2.1 have labels which correspond to the question parts below. As you answer each question part below, you can assume that your answers to the previous question part(s) have been implemented. For example, the answer to part (c) will be a collection of elements whose type was defined in the answer to part (b). For each of the question parts (a)-(e) below you should:

(i) Provide a separate code extract replicating the behaviour for the specific statements shown as the commented lines in Figure 2.1. For example, your answer for part (a) will consist of alternative code for the three statements commented with `//a`. Typically, your answer will involve new header file(s), declaration and executable statements. It is possible that as a consequence of your suggestion that some of the original statements will not be needed. Conversely you are free to introduce statements not in the original code of Figure 2.1.

(ii) Briefly describe the benefit of your suggested code selection.

(a) Figure 2.1 uses a C string library. Suggest and demonstrate another standard library for working with strings. [4]

(b) Figure 2.1 uses a user created data type for working with Complex Numbers. Suggest and demonstrate another standard library for working with Complex Numbers. [4]

(c) Figure 2.1 uses dynamic memory management for working with a variable sized array of data elements. Suggest and demonstrate another standard library for working with variable sized collections of data elements. [4]

(d) Figure 2.1 uses C Input/Output libraries. Suggest and demonstrate other standard libraries for Input/Output. [4]

(e) Figure 2.1 uses library conversion routines and Input/Output buffers for working with strings which contain embedded numeric data. Suggest and demonstrate other standard libraries for working with memory buffers. [6]

(f) Implementing the changes in (a)-(e) will lead to a more Object Oriented Program. Suggest and demonstrate a change to the code that could lead to a more Generic Program and briefly describe why this is beneficial. [4]

(g) List any other changes not already identified in (a)-(f) that demonstrate features and syntax available in C++ but not in C. [4]

```
/* Assume your header files appear here in a complete program */
#include <stdio.h>                                                    // d
#include <string.h>                                  // a
struct TComp{                                             // b
   double re; double im;                                  // b
};                                                        // b
typedef struct TComp TComplex;                            // b

#define TRUE 1
int main(){
  double re, im;
  int i;
  char s[100];                                   // a
  TComplex *z; /* i.e. array without dimension */ // c
  TComplex sum = {0.0,0.0};                      // b
  printf("How many complex numbers are to be entered? ");
  gets(s);
  int count = atoi(s);                                                // e
  z = (TComplex *) malloc(count*sizeof(TComplex));// c
  printf("Enter a pair of real and imag coeffs per line\n");
  for (i=0; i < count; i++){
     gets(s);
     sscanf(s,"%lf%lf", &re, &im);                                    // e
     z[i].re=re;                                        // c
     z[i].im=im;                                        // c
  }
  for (i=0; i < count; i++){
     sum.re = sum.re + z[i].re;                  // b
     sum.im = sum.im + z[i].im;                  // b
  }
  printf("Display results? ");                                  // d
  gets(s);                                                      // d
  int show = (strcmp(s,"yes") == 0);    // a
  if (show == TRUE){
     sprintf(s, "\nSum = (%3.1lf %3.1lf)", sum.re, sum.im); // e
     printf("%s\n", s);                                          // e
  }
  getchar();                                                    // d
}
```

Figure 2.1

```
How many complex numbers are to be entered? 2
Enter a pair of real and imag coefficients per line
1.1 2.2
3.3 4.4
Display results? yes
Sum = (4.4 6.6)
```

Figure 2.2

3. (a) (i) What does it mean to say that the UML provides Orthogonal views of a software system?

(ii) Why are Orthogonal views needed for developing software? Give an example from any other technical area (i.e. not Software Engineering) that also uses Orthogonal views. [6]

(b) Using only the class diagram in Figure 3.1(a) describe a `FullAdder` [2]

(c) In Figure 3.1(a) if the filled-in diamonds were replaced by empty diamonds, would the information conveyed by the diagram change? [2]

(d) Examine Figure 3.1(a) and each of the three object diagrams in Figures 3.1(b)-(d). For each object diagram state whether it is valid with respect to the class diagram and justify your answers. [3]

(e) Examine the six class diagrams labelled Figure 3.2(a)-(f). For each of the textual descriptions below state which is the equivalent class diagram or if you think none of the diagrams are applicable then state "None".

(i) `BottomUpParser` or `TopDownParser` are both kinds of `Parser`.

(ii) A `Scanner` and a `Parser` are associated by both having a `currentToken` and a `semanticValue`.

(iii) A `key` is used to look-up a value in a `SymbolTable`.

(iv) A `Production` in a grammar is recursive.

(v) A `PushDownAutomatom` consists of a `FiniteStateAutomatom` and a `Stack`. [5]

(f) Describe the main relationships between classes that are shown in a UML class diagram. Drawing upon any area of your undergraduate studies, draw a UML diagram demonstrating all the relationships. [6]

(g) A Moore FSA with state set `{Q0,Q1,Q2,Q3,Q4}`, input set `{a,b}` and output set `{0,1}` is described by the following state-transition and output tables. Draw the equivalent UML statechart (i.e. not State Transition Diagram) representing the same machine. [6]

	a	b
Q0	Q1	Q2
Q1	Q1	Q3
Q2	Q3	Q2
Q3	Q4	Q4
Q4	Q4	Q4

	Output
Q0	0
Q1	0
Q2	0
Q3	1
Q4	0

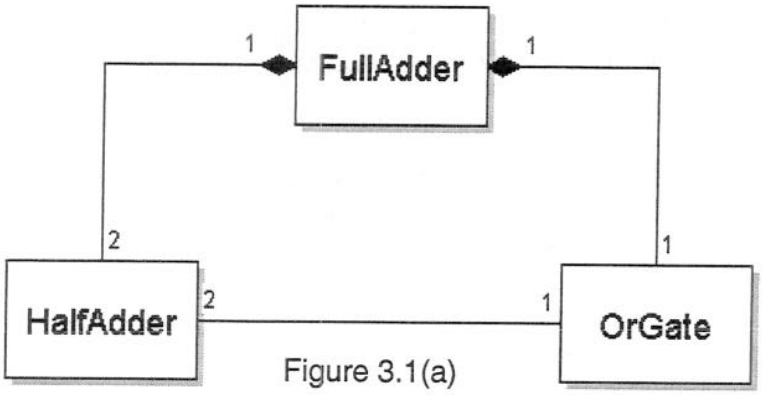


Figure 3.1(a)

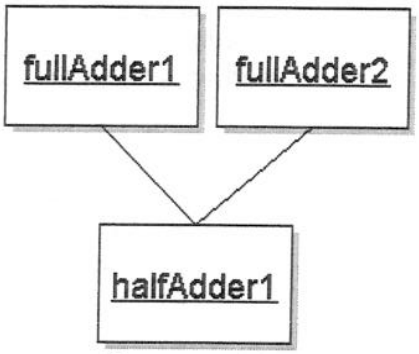


Figure 3.1(b)

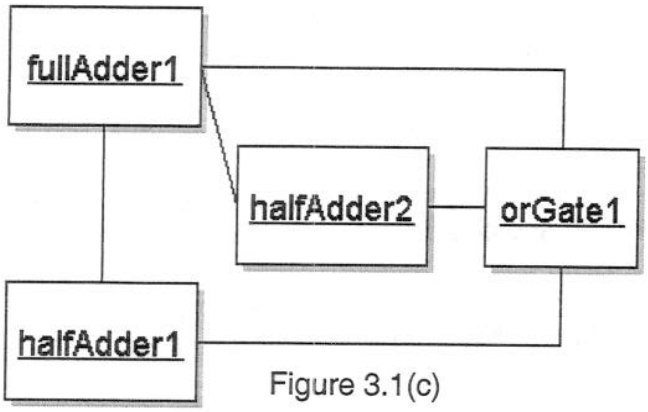


Figure 3.1(c)

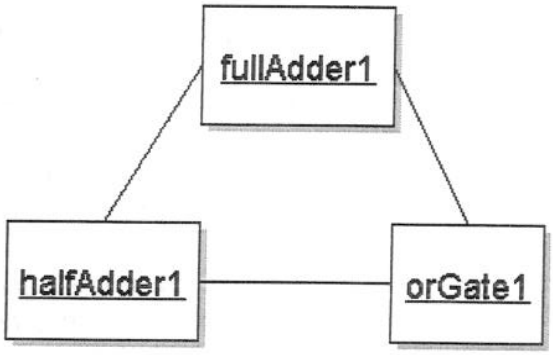


Figure 3.1(d)

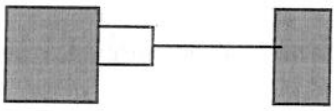


Figure 3.2(a)

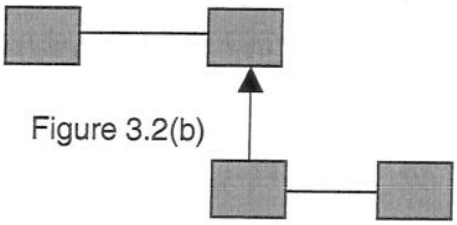


Figure 3.2(b)

Figure 3.2(c)

Figure 3.2(d)

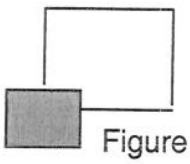


Figure 3.2 (e)

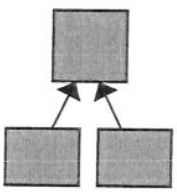


Figure 3.2(f)

4. This question concerns an event-driven Digital Logic Simulator (DLS). The bullet-points below describe a subset of the top-level functionality requirements for a software based DLS written in C++.

- Load a circuit description from a file and store the information in standard library container classes. Anticipate typical file errors.
- Initialise the *primary* input nodes of the circuit.
- Set a *primary* input logically high.
- Display all the nodes in the circuit.
- Simulate the circuit until the circuit settles.

(a) Using the requirements above, do the following:

(i) Draw a UML Use-case diagram for the software system.

(ii) Using any of the Use-cases identified in (i) write a complete textual description including one success and one failure scenario.

(iii) Draw a UML activity diagram (i.e. not a flowchart) representing a scenario (including success, failure or both) for *any other* Use-case i.e. that was not used for (ii).

(iv) Draw a UML state chart (i.e. not a State Transition Diagram) representing the states of the overall DLS software system.

(v) Describe how a truth-table for a Boolean function (e.g. an `and` gate) could be described in the Object Oriented terminology used in (a)(i) - (a)(ii) above.

[14]

(b) Perform a textual analysis on the following paragraph:

A circuit-file will include the definition of many gates and many nodes. A gate has a name, has a delay (measured in nanoseconds), has one output node and has one or more input nodes. Each gate evaluates some specific boolean function after the specified delay so the boolean function name is also recorded (i.e. e.g. and, nand etc...) A node has a name and has a logical state (i.e. high, low, undefined) An event represents a node state transition. The event records information including the state it will transition to, the time of transition and is associated with a specific node.

Based on the textual analysis make a list of the following:

(i) Filtered Candidate classes i.e. do not include rejected candidates.

(ii) Attributes of the classes identified in (b)(i) underlining associations.

(iii) Literal values of primitive attributes identified in (b)(ii).

(iv) Draw a UML class diagram based on your previous answers including only those details that are appropriate for a Conceptual Model. Include multiplicities and association names.

[8]

(c)

(i) Using your answer to question (b)(iv), select the one class in the UML class diagram that has the most attributes. Draw another UML class diagram but only for the chosen class and assume that this class diagram will form part of the Specification model.

(ii) For the class you have just selected, choose one simple behavioural aspect that demonstrates a Design Principle. Briefly describe the behaviour in words and specify which Design Principle is demonstrated and why?

(iii) Draw a UML Sequence Interaction Diagram (SID) demonstrating both the instantiation of an instance of the class chosen in (c)(i) and an interaction that involves the behaviour chosen in (c)(ii).

(iv) Translate your SID from (c)(iii) into one or two lines of C^{++} implementation code.

[8]

Final master - 8/6/07.

EE.12

Solutions 2007

A1 [Bookwork and New Computed example]

(a)

(i) a polymorphic variable e.g. `a1[0],a[1]` – will accept `a` and `TV::iterator i` [1]

(ii) a polymorphic member function e.g. `A::doit(),vector<T>::push_back()` [1]

(iii) a constructor or MF that has been overridden e.g. `A::doit()` [1]

(iv) a constructor or MF that has been overloaded e.g. `A::A(), A::A(char)` [1]

(v) a class constant e.g. `AB::DIGITS, string::npos` [1]

(vi) an abstract class e.g. `A` [1]

(b)

(i) a first class container class e.g. `vector` [1]

(ii) an overloaded operator for a standard library class e.g. `ostream::operator<<` - will accept `!= = -> ++ >=` [1]

(iii) a class that has a template data member e.g. `std::vector<T>` [1]

(iv) a member function that is found in all first class container classes e.g. `begin(),end()` [1]

(v) a non-mutating generic algorithm e.g. `std::string.find()` [1]

(vi) a pseudo-variable e.g. `this` [1]

(c)

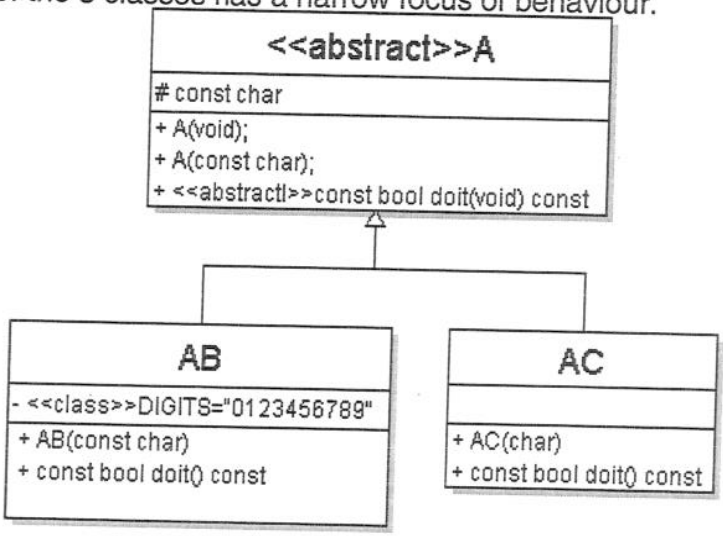
(i) Least Privilege i.e. don't grant any more privilege that is needed. Every manifestation of the keyword `const` is an example of its application. [3]

(ii) Information Hiding i.e. controls what the user "needs to know" and is "able to know". The former is manifest as the ADT/class and the latter by the visibility controls e.g. `private, protected and public.`
Alternatives: Use of iterators to hide the underlying data structure during iteration, Use of inheritance to hide protected members outside of logically related classes [3]

(iii) Separation of Concerns i.e. classes should model one kind of object and each method (behaviour) deals with exactly one well-defined small job e.g. `A::doit()` is a simple validator, `A::A()` associated with instantiation of an object. Each of the 3 classes has a narrow focus of behaviour. [3]

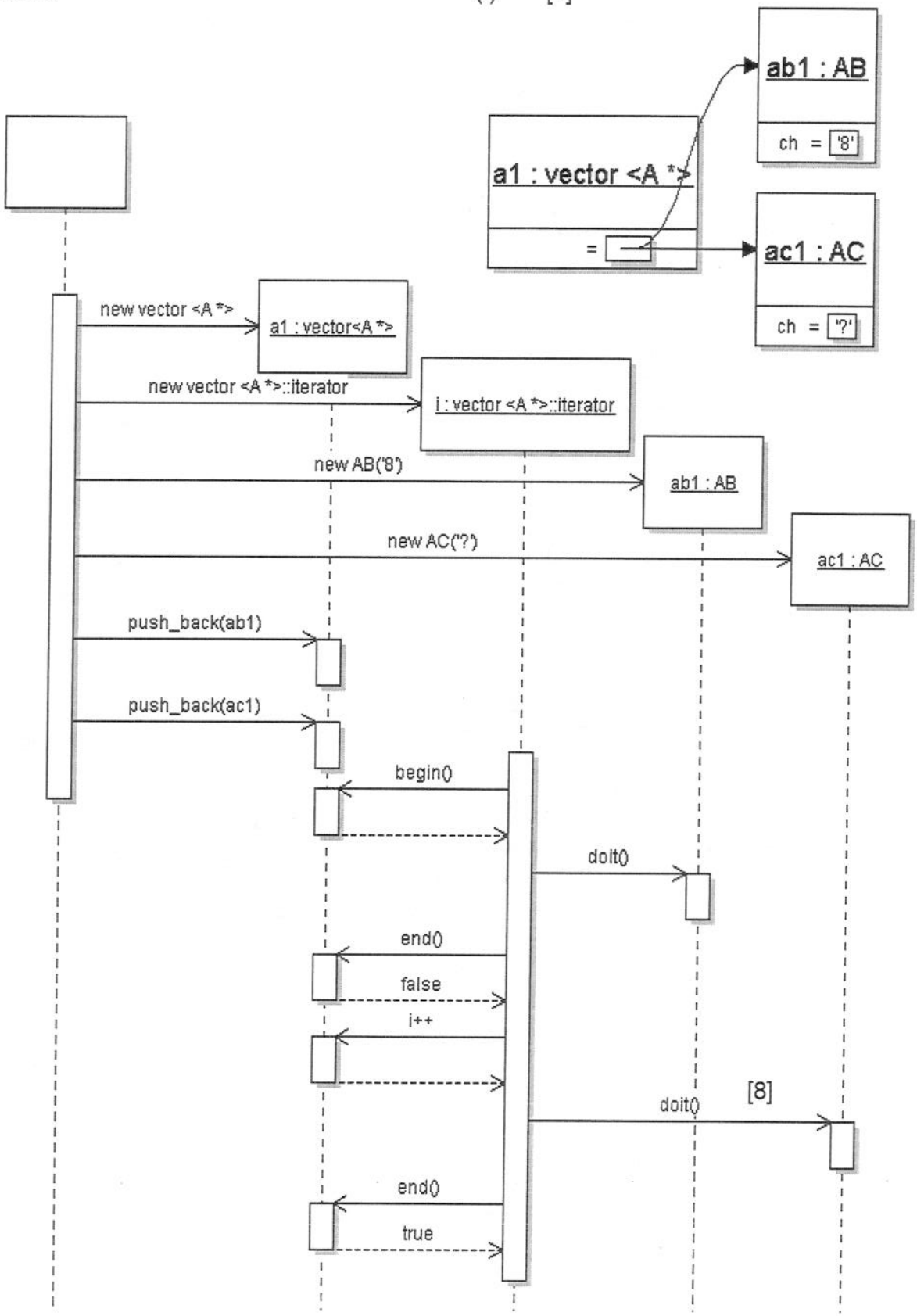
(d) true
false [1]

(e) [6]



A1. continued.

(f) [4]



(g) [8]

A2. [Bookwork and New Computed example]

(a)
```
#include <string>
string s;
inr show = (s.compare("yes")==0); // will accept operator==
```
Advantage(s): Simpler to use: don't need to worry about null-terminated,memory management [4]

(b)
```
#include <complex>
complex<double> sum(0.0,0.0);
sum = sum + complex<double>(z[i].re, z[i].im);
```
Advantage(s): robust working code, no need to re-invent the wheel [4]

(c)
```
#include <vector>
vector<complex<double> > z;
z2.push_back(complex<double>(re,im));
```
Advantage(s): Flexible, adaptable, robost component [4]

(d)
```
#include <iostream>
cout << "Display results? ";
getline(cin, s); // will accept cin >> s
cin.get();
```
Advantage(s): Generic producer/consumer concept, I/O manipiulators [4]

(e)
```
#include <sstream>
istringstream iss(s);
iss >> count;
istringstream iss(s);
iss >> re >> im;
ostringstream oss;
oss << endl << "Sum = " << sum << endl;
cout << oss.str();
```
Advantage(s): Integrates into streams [6]

(f)
```
vector<complex<double> >::iterator i;
for (i=z2.begin(); i != z2.end(); i++)
   sum2 = sum2 + *i;
```
Advantage(s): High level of abstraction hides underlying data-stucture [4]

(g) `#include <stdio.h>` => `#include <cstdio>` & `using namespace std;`
`#define TRUE 1` => `const int TRUE = 1` or use a bool literal in program
`int show = (strcmp(s,"yes") == 0);` =>
`bool show = (s.compare("yes")==0);`
`int main()` i.e. don't check args => `int main(void)` i.e. no args
Advantage(s): Adds clarity to code [4]

A3 [Bookwork and New Computed example]

(a) (i) A software system can be described in terms of its Functionality i.e. a top-level description of what it does, Structure i.e. what components/modules it is constructed from and how they are connected and Behaviour i.e. a bottom-up description of how some particular functionality is delivered. [3]

(ii) We need a structural description to build a software system and Functional/Behavioural descriptions in order to satisfy requirements. Thus we need all 3 views for a complete description of the software system.

A Civil Engineer building a bridge would have functional description e.g. carries road/rail, number of lanes of traffic. Structural description in terms of plan/view/elevation/electrical/plumbing diagrams and Behavioural description e.g. stress analysis of concrete supports, tensile strength of cable. [3]

(b) A Full-adder consists of 2 Half-adders and an Or-gate. Each Half-adder is associated with an Or-gate and each Or-gate is associated with both Half-adders. [2]

(c) Aggregation and Composition are particular associations where there is a whole/part-of association e.g. an Invoice is made up of instances of InvoiceLine, a Chessboard is made up of instances of Square, a Circuit is constructed from instances of Component..

The Composition association is a stronger form of the "is-part-of" relationship and is denoted by the closed diamond at the 'whole" end. If a part is missing then the whole is incomplete, for example a Chessboard with less than 64 instances of Square would be incomplete.

The Aggregation association is also an "is-part-of" relationship and is denoted by the open diamond at the 'whole" end e.g. Component *isPartOf* Circuit i.e. a circuit may have a variable number of components, for example we might replace a number of OR and AND gates providing Half-Adder functionality with a single Half-adder component. Thus the information conveyed is changed, since it is a less strong form of the whole/part-of association. [2]

A3 continued.

(d) (i) Invalid because each HA is associated with a maximum of 1 FA, not 2.

(ii) Valid because each FA is associated with 2 HA's and 1 Or. Each HA is associated with 1 FA and 1 OR. Each OR is associated with 1 FA and 2 HA's. The invariants expressed by the multiplicities of the associations have not been violated.

(iii) Invalid because each FA should be associated with 2 HA's, not 1. [3]

(e) (i) `BottomUp` and `TopDown` are both kinds of a `Parser`. Fig 3.2(f)

(ii) A `Scanner` and a `Parser` share a `token` and `semantic value`. None

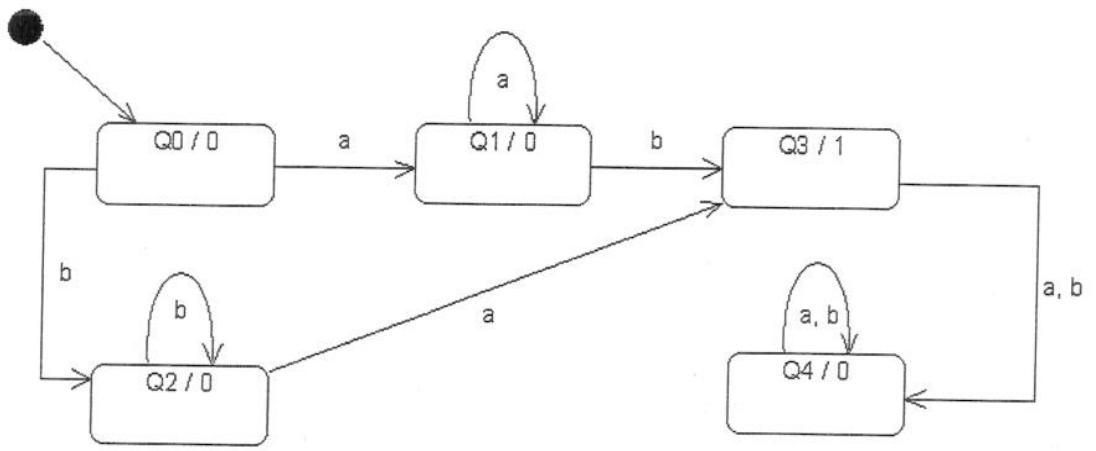
(iii) A `key` is used to look-up a value in a `SymbolTable`. Fig 3.2(a)

(iv) A `Production` in a grammer may be recursive. Fig 3.2(e)

(v) A `PushDownAutomatum` consists of an `FiniteStateAutomatum` and a `Stack`. Fig 3.2(c) [5]

(f) Association/Generalisation and Dependency.
Possible UML class diagrams: Analogue Filter, Digital Logic Simulator, Arithmetic/Geometric Series, Polar/Cartesian Vectors. [6]

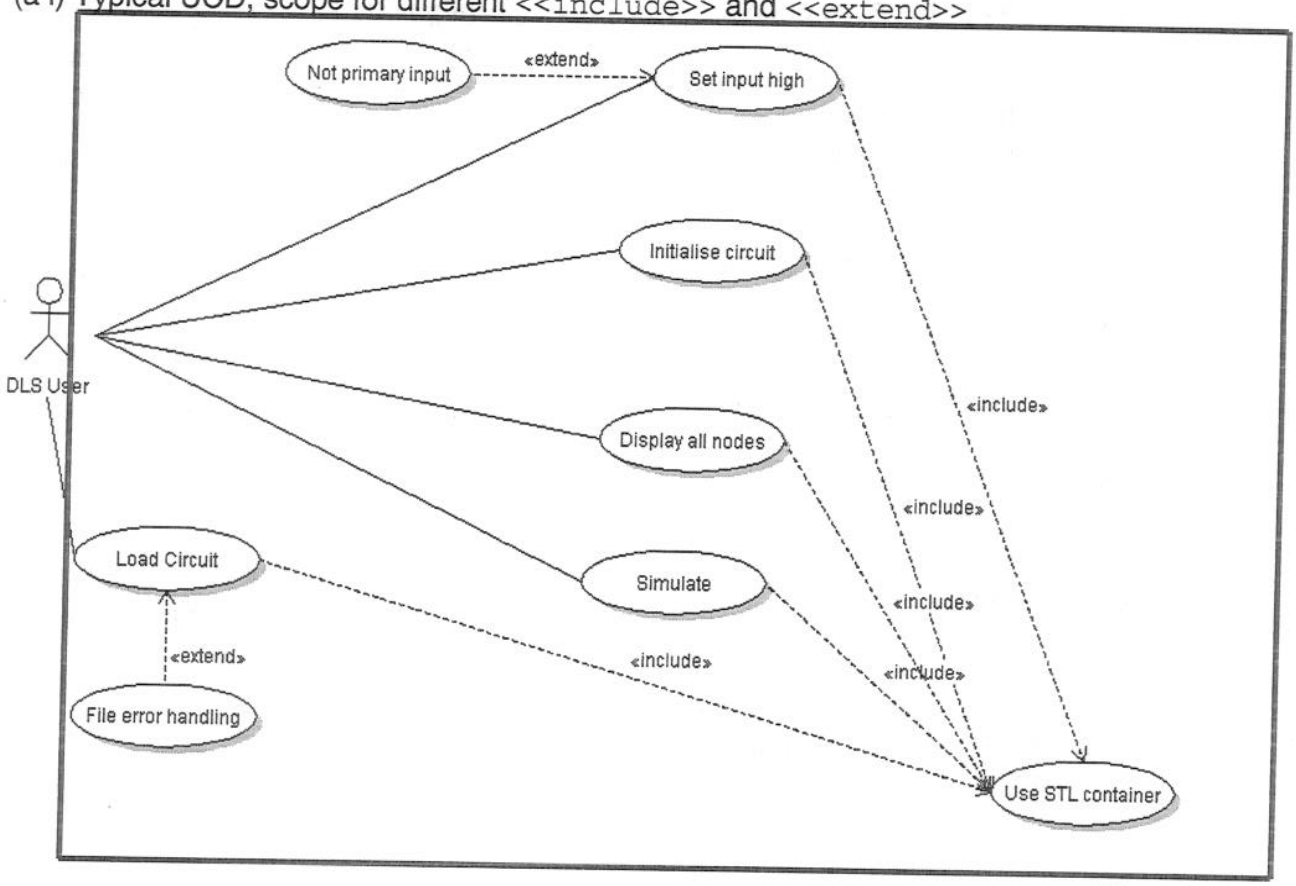
(g)



[6]

A4. [Bookwork and New Computed example]

(a i) Typical UCD, scope for different <<include>> and <<extend>>



[3]

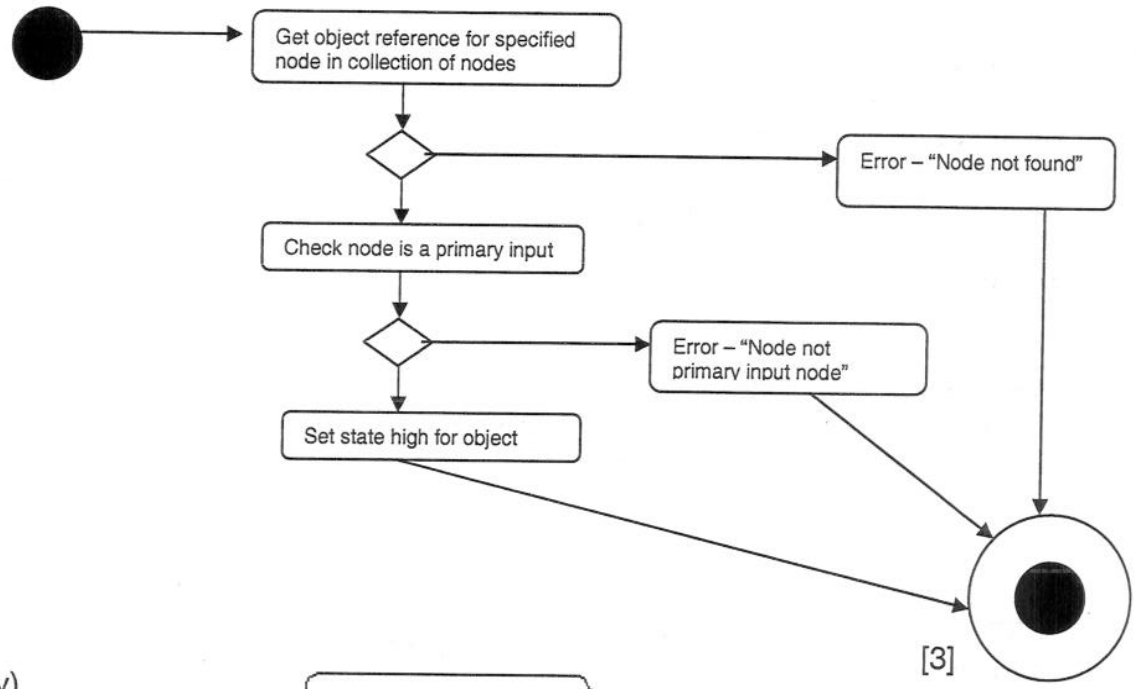
(a ii) e.g.

Identifier and name: UC#1, Set input high
Initiator: DLS User
Goal: To set a primary input logically high
Pre-condition: Node must exist, node must be primary
Post-condition: Primary input node is high

	DLS user	System
Success scenario	`H a`	
		Get object reference for node `a` in collection of nodes
		Check node is a primary input
		Set state high for object
Failure scenario	`H z`	
		Get object reference for node `z` in collection of nodes
		Error because node `z` not found

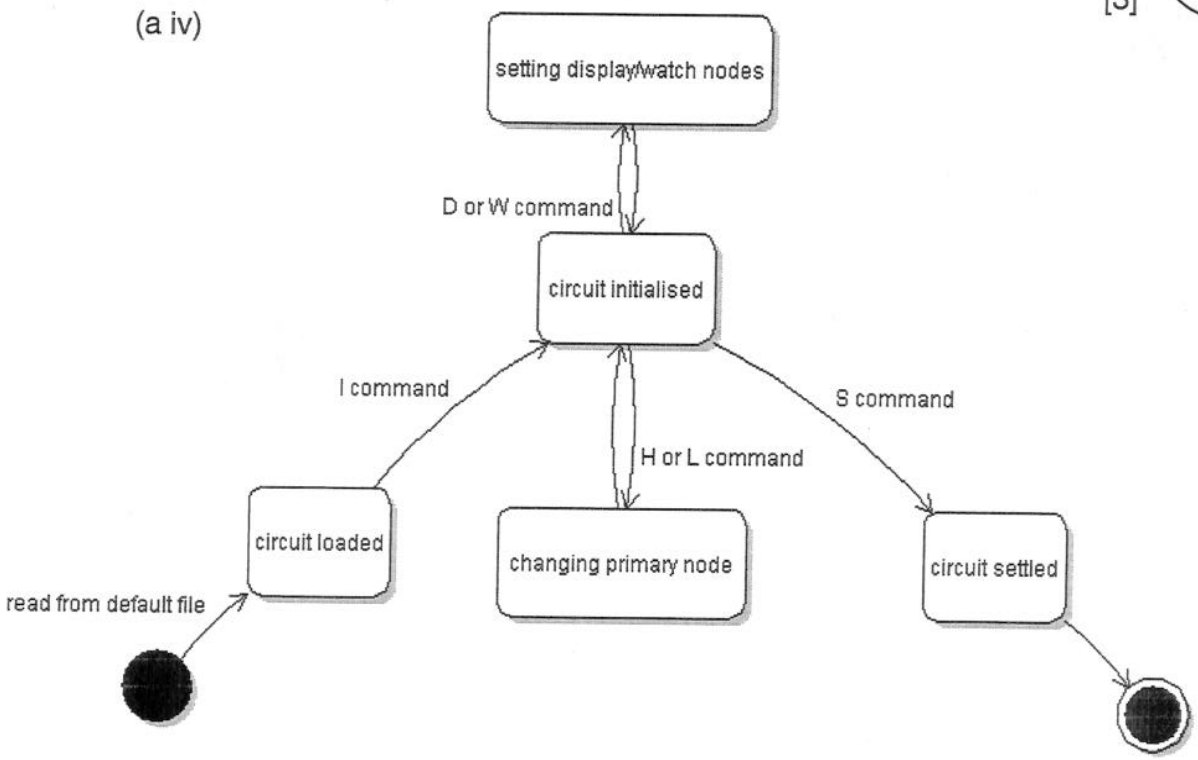
[3]

A4 continued. (a iii) Set input high scenarios



[3]

(a iv)



[3]

(a v) Each of row of truth table is equivalent to a success scenario.
The whole truth table is equivalent to a Use case. [2]

A4 continued.

(b i) Candidate classes. `CircuitFile, Gate, Node, Event` [1]

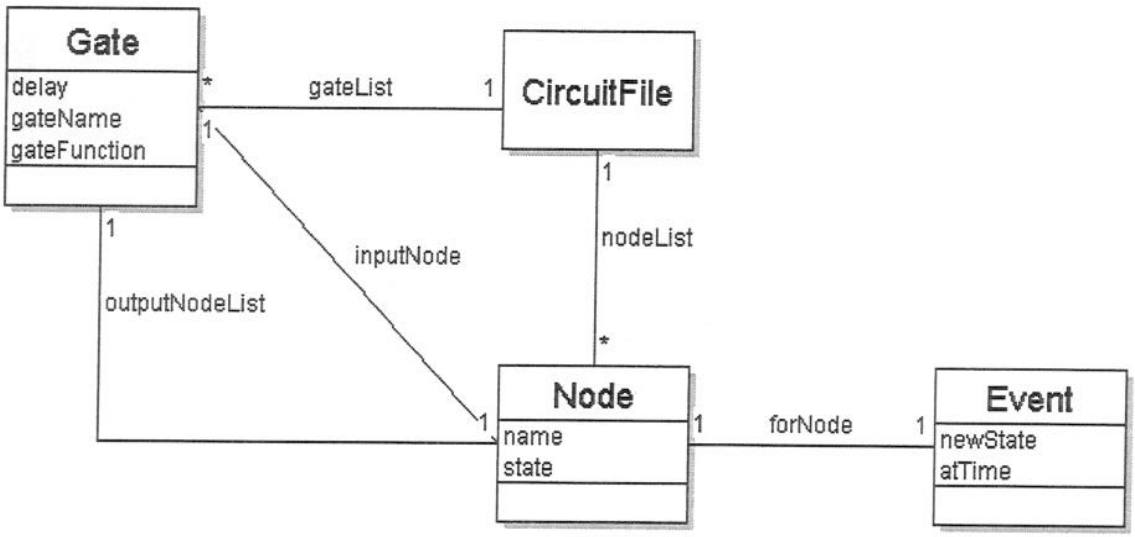
(b ii) Attributes of classes in (i).

`CircuitFile(gateList, nodeList)`
`Gate(name, delay, outputNode, inputNodeList)`
`Node(name, state),`
`Event(newState, atTime, forNode)` [2]

(b iii) Literal values of attributes identified in (ii).

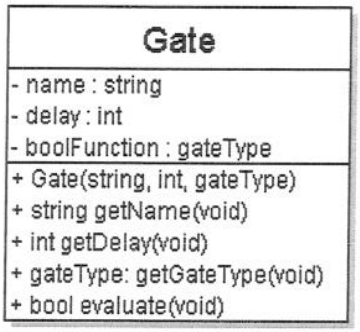
`gateFunction (and, nand, or, nor, exor)`
`state (high,low,undefined)` [1]

(b iv)



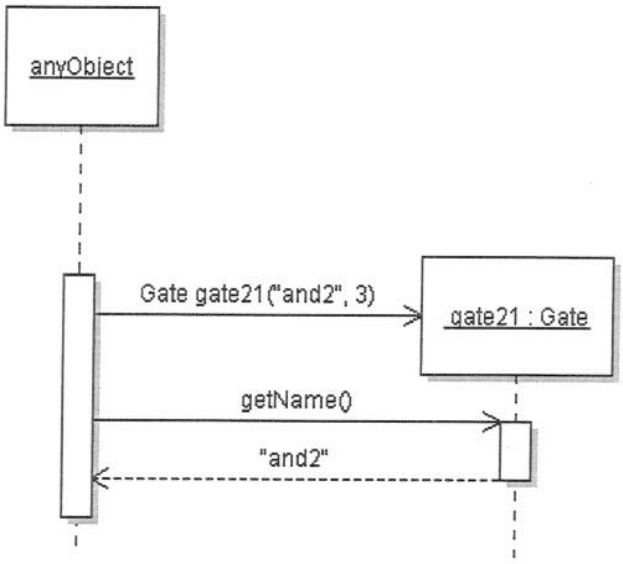
CM should not contain: Navigation arrows, Member functions, Datatypes. [4]

A4 continued. (c i) [2]



(c ii) `getName()` returns the value of data member that records the name of the gate and is a getter method. Respects the Design Principle of Separation of Concerns i.e. a Member Function should do one think and only one thing. [2]

(c iii)



[2]

(c iv)
```
Gate gate21("and2", 3);
string s = t.getName()
```
[2]