ENGINEERING TRIPOS PART IIA

Thursday 12 May 2005 2.30 to 4

Module 3F6

SOFTWARE ENGINEERING AND DESIGN

Answer not more than ***three*** *questions.*

All questions carry the same number of marks.

The ***approximate*** *percentage of marks allocated to each part of a question is indicated in the right margin.*

There are no attachments.

You may not start to read the questions printed on the subsequent pages of this question paper until instructed that you may do so by the Invigilator

(TURN OVER

1 (a) Software testing is concerned with verifying that a piece of software conforms to its specifications. What is the difference between *defect testing* and *statistical testing*? Do these verify conformance with the software's *functional* or *non-functional* requirements? [15%]

(b) Figure 1 shows a function to calculate the value of a real-valued number raised to an integer power, i.e. `exp(a,b)` $\equiv a^b$.

(i) What is *functional* (black-box) testing, and how does it make use of *equivalance partitioning*? Suggest five appropriate test cases (i.e. values of `a` and `b`) to test the behaviour of the function given in Fig. 1. [20%]

(ii) What is *structural* (white-box) testing? Draw a program flow graph for the function of Fig. 1, and suggest a minimal set of test cases which would ensure that every path has been covered. [30%]

(iii) What is *software inspection* (static testing), and how is it a form of software verification? How useful is it as a means of detecting program errors? [15%]

(c) As written, the function will fail on some inputs. What is the failure and which of the three methods described in section (b) are likely to detect this failure? [20%]

(cont.

```
float exp(float a, int b)
{
   float ans = 1.0;
   if(b < 0)
   {
      b = -b;
      a = 1.0 / a;
   }
   while(b > 0)
   {
      ans = ans * a;
      b = b - 1;
   }
   return ans;
}
```

Fig. 1

2 Figure 2 shows some C++ code, written in a *function-oriented* style, to push (add) and pop (remove) integers stored on a first-in-last-out stack. It is desired to make this code (i) object-oriented and (ii) able to cope with concurrent operations on the stack. Assume that the two exceptions used in Fig. 2 have already been defined elsewhere.

(a) Considering the issues of *coupling* and *coherence*, discuss how object-oriented programming helps the writing of code which is easy to maintain. [20%]

(b) *Monitors* are an object-oriented construction which provide a good solution to concurrency problems. Describe how monitors can provide both mutual exclusion and synchronisation. [15%]

(c) Rewrite the code of Fig. 2 as a monitor object, showing both the monitor declaration and the member function definitions. (Assume that the `monitor` keyword is available in this version of C++, and, apart from the concurrency support it provides, is used in the same way as `class`.) [25%]

(d) It is now necessary to make the stack object available to other users across a network, using CORBA. Write a CORBA IDL file to define the interface to your monitor object. [10%]

(e) As it stands, the monitor object will throw exceptions if the user attempts to push a value onto a full stack, or pop a value from an empty stack. It is suggested that the CORBA interface should be extended to allow the clients to query the current number of values on the stack. Show how the IDL file can be modified to do this, without breaking any existing clients. [10%]

(f) It is now decided that, instead of throwing exceptions, the push and pop member functions should block until the stack is in a valid state. Modify the push and pop member functions to achieve this, making use of condition variables. [20%]

(cont.

```
const int max_size = 100;
int stack[max_size];
int stackpos = 0;

void push(int a)
{
   if(stackpos == max_size)
      throw StackOverflowError;

   stack[stackpos] = a;
   stackpos = stackpos + 1;
}

void pop(int& a)
{
   if(stackpos == 0)
      throw StackEmptyError;

   stackpos = stackpos - 1;
   a = stack[stackpos];
}
```

Fig. 2

3 (a) Object oriented design permits the use of class derivation and polymorphism (virtual functions). Give a brief description of each of these facilities. [20%]

(b) A supermarket is commissioning a software system to perform stock control. Part of the specification for the software reads as follows:

Specification: The supermarket has many sections (e.g. bakery, diary, fruit & vegetables etc.). Each section controls several stock lines (e.g. the bakery has a bread stock line and a cake stock line). Each stock line contains several products (e.g. the bread stock line has baguette, high-tin, bloomer etc.). The system must keep track of how many items of each product type are in the supermarket and their prices. Products fall into one of the following categories (frozen, refrigerated or durable). The supermarket has a number of shelves. Each product is kept on a particular shelf. When a customer purchases their products at the checkout, the system must update its record of the number of each product on the shelves. A watchpoint can be placed on the stock level of a product. When the number of items of a product falls below a level specified by the watchpoint, an order is sent to the distribution centre for replacement of that item. Orders for different categories of products must be placed differently (e.g. an order for a frozen product must be made to a different distribution centre from an order for a durable product).

Using good design principles, draw a UML class diagram which shows the main classes that will be needed for this software, the relationships between these classes and the main attributes and operations that they should support. Identify an example of polymorphism used in your design. You do not need to give any pseudocode in your answer. [55%]

(cont.

(c) The supermarket now wishes to extend the software to allow special offers on various products. There are two kinds of special offers; the first gives a discount on the price of the product, while the second gives additional loyalty points to the customer.

Show what changes will be required to the software in order to support this additional functionality, drawing any modified parts of your class diagram and identifying any design patterns used. [25%]

4 Figure 3 shows a UML class diagram for software which provides a control system for a car.

(a) Identify the classes present and the relationships between them. You do not need to describe any attributes or operations. [15%]

(b) Figure 4 shows C++ code that creates a simple control system for the fuel injector of a car. Draw a sequence diagram which shows what happens when the `do_control` method is called. [35%]

(c) If the components were added to the control system in a different order, would the behaviour of the control system be affected? If the driver makes a change to the position of the accelerator pedal, how many times must the `do_control` method be called before there is a change in the amount of fuel being delivered to the engine? Explain your answers. [25%]

(d) It is now desired to use the control system in several different types of car. Unfortunately the hardware in each type of car is different and requires different interface software. Thus, for example, the `Accelerator` class designed for one type of car cannot be used with another type of car.

Using good design principles, show the necessary changes to the software to allow this, for the case of one additional car type, drawing any modified parts of the class diagram, and identifying any design patterns used. You do not need to give any pseudocode in your answer. [25%]

(cont.

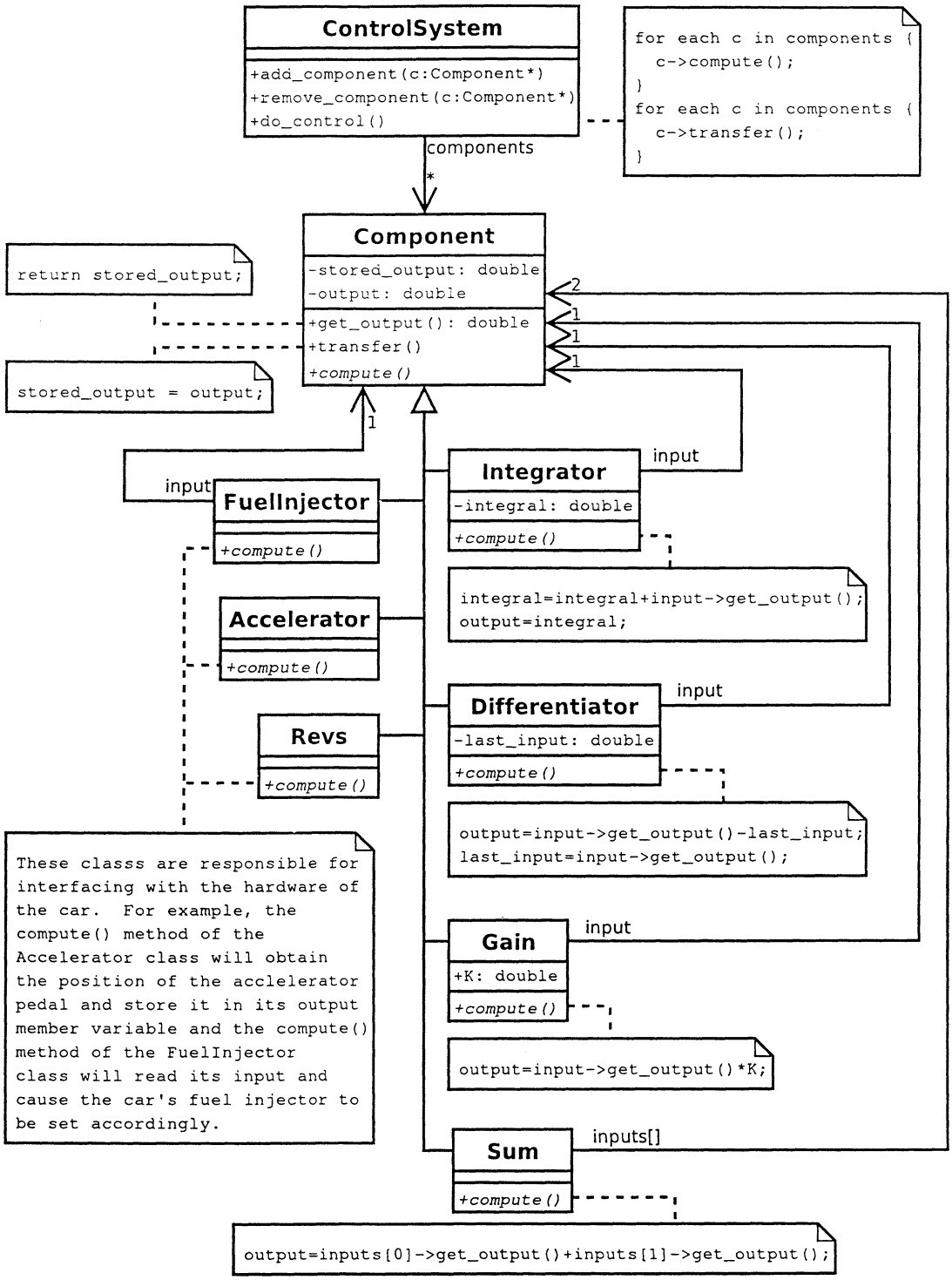

Fig. 3

(CONTINUED OVER.

```
int main()
{
   ControlSystem cs;

   Accelerator a;
   cs.add_component(&a);

   Differentiator d;
   d.input = &a;
   cs.add_component(&d);

   Sum s;
   s.inputs[0] = &a;
   s.inputs[1] = &d;
   cs.add_component(&s);

   FuelInjector f;
   f.input=&s;
   cs.add_component(&f);

   while(true)
   {
      cs.do_control();
   }
}
```

Fig. 4

END OF PAPER