SOFTWARE ENGINEERING 2: OBJECT ORIENTED SOFTWARE ENGINEERING

1. This is a general question about C++ and Object Oriented Software Engineering.

   a) Consider an application domain related to wheeled vehicles, in particular dealing with bicycles and cars. Bicycles have a front and a back wheel. Cars have two front wheels and two back wheels. For all wheeled vehicles we want to be able to know the cost of their set of wheels (as the sum of the cost of each of their wheels).

      i) Describe in words how you would model this domain in an object oriented architecture.

[8]

[bookwork, new example]

For this domain, concrete classes could be modelled for each of the wheeled vehicles, `Bicycle` and `Car`, as well as their component `Wheels`. The `Bicycle` and `Car` classes should inherit from an abstract class `WheeledVehicle`. A single method `get_cost` would be defined in `Wheel`. A method for computing a wheeled vehicle's estimated cost, `get_wheels_cost` should be declared as virtual in the base class `WheeledVehicle`, such that the two derived classes would override this operation.

Each of the derived classes will be represented by multiple `Wheels` (composition), namely four for the `Car` class and two for a `Bicycle`.

Every `Wheel` is represented by its cost and relies on this property to determine the overall cost of the wheel. In order to acquire the cost of a subtype of `WheeledVehicle`, delegation to the components' cost estimation operations would be required. A `Bicycle` would return the aggregate cost of its front and back wheels, whilst a `Car` would return the total cost of its four wheels.

[Most students described the overall architecture correctly and mentioned at least some of its elements in terms of OOP. Some did not name the relationships of inheritance and composition.]

      ii) Draw a UML class diagram of the architecture.

[8]

[bookwork, new example]

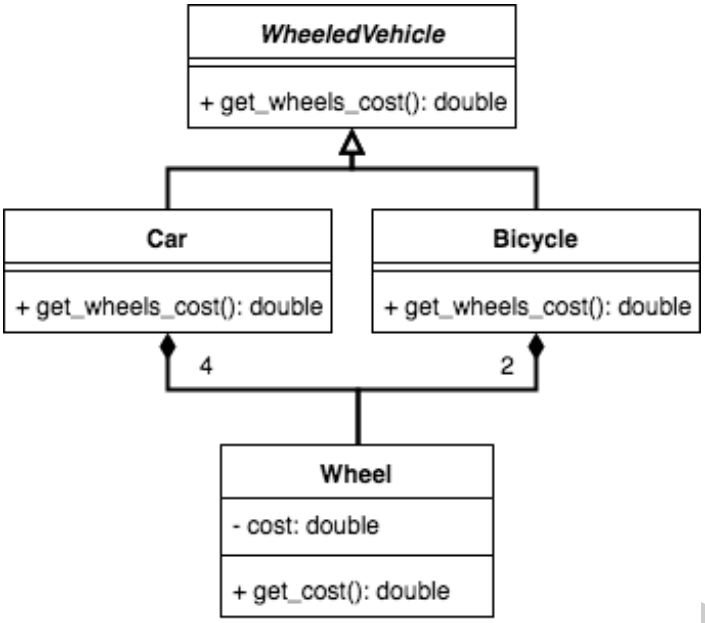


Figure 1.1 Often attributes that are involved in composition relationships are not also included in the class box: the diamond arrow is annotated instead. Either way is considered acceptable in this case. The get_cost and get_wheels_cost functions are not marked as const in the diagram as this aspect is primarily part of the implementation rather than the design, but this information may be included too. The base class should be denoted as abstract in some recognisable way, it can be done also by including an annotation.

[Most students included most components of the diagram correctly.]

iii) Write a declaration for all the classes. The declarations can be kept to the essential skeleton (e.g. constructors can be omitted) but all the relevant elements needed in order to express the architecture should be included. Include appropriate implementation(s) for the operation related to the cost of the set of wheels of wheeled vehicles.

[10]

[bookwork, new example, programming]

```
class WheeledVehicle {
    public:
        virtual double get_wheels_cost() const = 0;
};
```

[2]

```
class Car : public WheeledVehicle {
    public:
        double get_wheels_cost() const{
            return front_right.get_cost()
            + front_left.get_cost()
            + rear_right.get_cost()
            + rear_left.get_cost();
```

```
        }
    private:
        Wheel front_right;
        Wheel front_left;
        Wheel rear_right;
        Wheel rear_left;
};
```

[3]

```
class Bicycle : public WheeledVehicle {
    public:
        double get_wheels_cost() const{
            return front.get_cost() + back.get_cost();
        }
    private:
        Wheel front;
        Wheel back;
};
```

[3]

```
class Wheel {
    public:
        double get_cost() const{
            return cost;
        }
    private:
        double cost;
};
```

[2]

[Most students wrote most of the code correctly. Some did not write a correct implementation of the function returning the cost of the wheels in the vehicle classes, for example erroneously directly accessing the private member data of the class representing the wheel.]

b) Explain the concept of function binding, distinguishing between "early" and "late" binding. Discuss how this concept relates to polymorphism in C++, illustrating your answer with an example in C++ code.

[8]

[bookwork, programming]

Binding refers to the process in which a function call is matched correctly with a function definition.

Static (or early) binding is the default feature in C++, whereby the compiler and linker connect the function call with its body at compile time. Dynamic (or late) binding instead resolves the function call at runtime when the program is actually executed.

Polymorphism is the principle of providing a single interface to multiple subtypes via a base class. Late binding is required for polymorphism, which in C++ is enabled using the keyword *virtual*.

For example, consider an operation `func` that prints "Base" for a `Base` object and "Derived" for `Derived` objects:

```
class Base {
    public:
        virtual std::string func() const{
            return "Base";
        }
};

class Derived : public Base {
    public:
        std::string func() const{
            return "Derived";
        }
};

int main(){
    Base* bptr;
    char type;

    std::cout << "Type choice (b/d) = ";
    std::cin >> type;

    if(type == 'b'){
        bptr = new Base();
    }
    else if(type == 'd'){
        bptr = new Derived();
    }

    std::cout << "Selected type: " << bptr->func() << std::endl;

    return 0;
}
```

By declaring the method `func` as virtual, the correct operation will be called on a `Derived` instance through a pointer or reference to the `Base` class.

[Many students answered including at least some correct ideas on what function binding is and on the difference between 'early' and 'late' binding. A few did not include a clear or valid example showing polymorphic behaviour. For instance in some cases the example didn't make use of any pointers or references.]

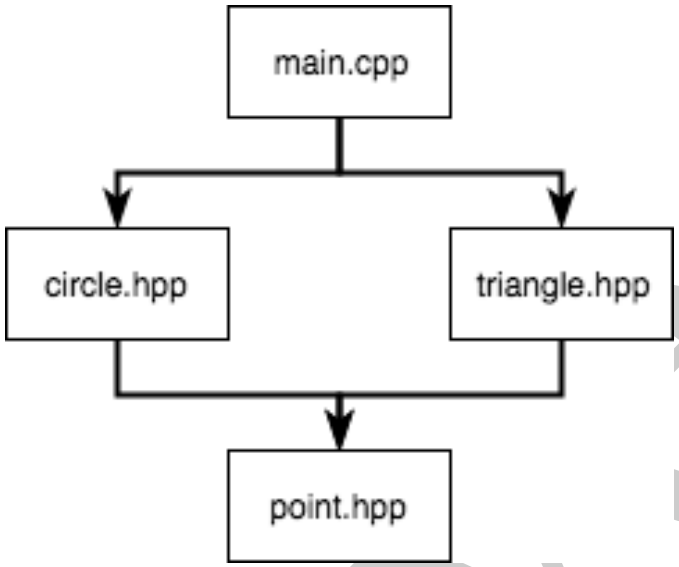
c) Explain what are `#include` guards in C++ header files and why they are important. Illustrate your answer with an example.

[6]

[bookwork, example analysis]

Include guards are constructs added to header files so as to prevent multiple definitions of any identifiers existing in a header file when it is included in other files via an `#include` directive.

Consider the following graph tracking the dependencies for a program handling geometric entities, such as `Triangles` and `Circles`.



In the geometric entities main source file, class `Point` is included twice through the *triangle.hpp* and *circle.hpp* inclusion, therefore resulting in a compilation error. The `Point` class should contain the following guard in its class declaration file in order to avoid this re-definition problem:

```
#ifndef POINT_HPP
#define POINT_HPP
class Point {
    ...
};
#endif
```

The `#ifndef` directive will now prevent multiple inclusions of class `Point` by omitting the content of the header file when `POINT_HPP` has already been defined during the first inclusion.

[Many students answered correctly. A few did not answer on the topic of guards but just on header inclusion in general (and without mentioning or addressing the multiple inclusions issue).]

2. This question deals with C++ templates and the Standard Template Library.

   a) i) Write a C++ template function that swaps the contents of two objects of the same generic type.

[4]

[bookwork, programming]

```
template <typename T>
void my_swap(T& a, T& b){
    T c = a;
    a = b;
    b = c;
}
```

[Most students answered correctly. Some did not pass the parameters by reference.]

ii) Using the swap function defined above, define another function that takes in input a `std::vector` containing elements of a generic type and changes it so that after the function call its elements are in reversed order. The implementation must make use of iterators.

[9]

[bookwork, programming]

```
template <typename T>
void reverse_vector(std::vector<T>& vec){

    typename std::vector<T>::iterator begin = vec.begin();
    typename std::vector<T>::iterator end = vec.end();

    if(begin != end){
        --end;

        while(begin < end) {
            my_swap(*begin, *end);
            ++begin;
            --end;
        }

    }
}
```

[Many students got the general idea right. Some didn't write the loop conditions correctly. Some didn't use the correct syntax for the declaration and/or dereferencing of iterators.]

b) Explain why the usual separation of function declarations from their corresponding definitions in separate header and source files is not possible with template functions.

[7]

[bookwork]

A template function is not a complete implementation but only a template for the compiler to use in order to create an actual instance of the function.

The compiler generates the needed specialisation of the template function when it encounters a function call with the actual parameter types.

At the point of instantiation, the compiler requires the full template function definition in order to perform all the syntactic checks on the provided types and generate the code for the actual instance.

This happens at compile time when each source file is compiled separately (before the linking).

As a result, you cannot separate the definition of a template function from its declaration.

To get around this problem, the definition of a template should be included in the same header file as its declaration.

[Many students included at least some of the relevant elements of the answer.]

c) Write C++ code for a template class `MyComplex` class that can be used to instantiate and operate on complex numbers. The class should include:

- A constructor with two generic parameters of the same type to represent the complex number's real and imaginary parts.
- A friend function that overloads the insertion (<<) operator for printing complex numbers.

[10]

[bookwork, programming]

```
template <typename T>
class My_Complex{
    public:
      My_Complex(T i_r, T i_img) : real(i_r), img(i_img){}

      template<typename T2>
      friend std::ostream& operator<<(std::ostream& out,
            const My_Complex<T2>& c);
    private:
      T real;
      T img;
};

template <typename T>
std::ostream& operator<<(std::ostream& out,
            const My_Complex<T>& c){
    out << "(" << c.real << ", " << c.img << ")";
    return out;
}
```

[Most students wrote correct code for the class declaration and the constructor. A few did not use the correct syntax for the overloading of the insertion operator, for example not declaring the type of the parameter as a generic type.]

3. This question deals with C++ exceptions.

a) What is "exception safe" code?

[4]

[bookwork]

Exception safe code keeps the state consistent and handles resources suitably even if an exception is thrown.

[Many students answered correctly. Some provided vague or incomplete answers.]

b) Consider the following function:

```
int min_value(const std::vector<int>& v);
```

Which returns the minimum value contained in the input vector. Discuss why and how the use of exceptions might be appropriate in this case.

[6]

[bookwork, example analysis]

If the vector in input is empty there is no meaningful return value for this function. We don't have the option of an "error code" return value either because there are no values which we can exclude from a normal return of the function. We have the option of considering an empty vector in input as undefined behaviour for the function. Or we could throw an exception if the vector is empty.

[Many students mentioned the salient aspect of this question as part of their answer.]

c) A function contains an `exit` instruction that is executed when something is not right. Discuss why and how the use of exceptions might be appropriate in this case.

[6]

[bookwork, example analysis]

The `exit` instruction causes the entire program (not just the function) to terminate and it conveys only limited information such as an error code to the operating system. If the function instead throws an exception, this can be caught and handled in the caller. Moreover different kinds of exceptions can be thrown depending on what situation arises so that distinct operations

can be performed by the exception handling in the catch block in the caller.

[Many students correctly mentioned the issue with using the `exit` instruction. Some did not describe an appropriate alternative using exceptions.]

d) Consider the following code. Trace its execution and its output including an explanation for all the steps.

```
#include <iostream>
#include <string>
#include <exception>

class class_a{
    public:

    class_a(int i_n) : n(i_n){
        std::cout << "a" << n << std::endl;
    }

    void f(){
        std::cout << "f1" << n << std::endl;
        std::string s = "z";
        throw s;
        std::cout << "f2" << n << std::endl;
    }

    ~class_a(){
        std::cout << "t" << n << std::endl;
    }

    private:
        int n;
 };

int main(){
    class_a c1(8);

    try{
        class_a c2(9);
        std::cout << "1" << std::endl;
        c1.f();
        std::cout << "2" << std::endl;
        std::exception e;
        throw e;
        std::cout << "3" << std::endl;
    }
    catch(const std::exception& e){
        std::cout << "e" << std::endl;
    }
    catch(const std::string& msg){
        std::cout << msg << std::endl;
```

```
    }

    std::cout << "3" << std::endl;

    return 0;
}
```

[14]

[bookwork, example analysis]

The program would generate the following output:

```
a8
a9
1
f18
t9
z
3
t8
```

Initially ‘a8’ is printed by the constructor of c1 (because it is constructed assigning 8 to its member data n) then ‘a9’ by the constructor of c2 (because it is constructed assigning 9 to its member data n) and the following instruction prints ‘1’. [1.5]

Then function f is called on c1, this prints ‘f1’ followed by the value of n in c1, which is 8, so it prints ‘f18’. [2]

The next instruction throws an exception as std::string, therefore the execution of the function is interrupted and also the rest of the instructions in the try block aren’t executed. [3]

However the destructor of c2 (which is declared inside the try block) is called and this prints ‘t9’. [2.5]

The execution then goes to the catch blocks. The exception thrown is not of type exception but it’s of type string so the catch for string is executed and ‘z’ (the content of the string) is printed. [2.5]

Then the instruction printing ‘3’ is executed and finally the destructor for c1 is called which prints t8. [2.5]

[Many students described some parts of the execution correctly. Some did not describe the correct flow of execution after the exception is thrown. Some did not include the destructor(s) being executed or did not place them at the correct point.]