Software Development with C++ – Exam Revision

Multiple Choice – 5 Questions

- Circle the correct answer, there is one answer per question. Each question is worth 1 mark.

1. Which of the following would be a valid functor declaration that takes two floats and returns an integer?
   a) int functor() (float a, float b);
   b) int operator() (float a, float b);
   c) int operator*() (float a, float b);
   d) int object() (float a, float b);

B

2. How do you cast complex objects to their base class in C++?
   a) static_cast<ChildClass>(new ChildClass{})
   b) dynamic_cast<ChildClass *>(new BaseClass{})
   c) dynamic_cast<BaseClass *>(new ChildClass{})
   d) dynamic_cast<BaseClass>(ChildClass{})

C

3. How do you inherit public members from a base class in C++?
   a) public ChildClass implements BaseClass
   b) public ChildClass : BaseClass
   c) public class ChildClass : public BaseClass
   d) public class ChildClass : BaseClass

C

4. What of the following are correct iterator types?
   a) Input, Forward, Bidirectional, Random access iterators
   b) Backward and forward iterators
   c) Output, Input, Forward, Backward, Bidirectional, Random access iterators
   d) Output, Input, Forward, Backward, Bidirectional, Random access iterators

A

5. What of the following are creational design patterns?
   a) Creation, Adapter, Factory
   b) Adapter, Proxy, Type-Object, Singleton
   c) Prototype, Builder, Factory, Decorator
   d) Builder, Factory, Singleton, Prototype

D

Short Answer – 5 Questions

1. What are the components of the builder design pattern?
    - Director
    - Builder – abstract class
    - Concrete Builder
    - Product
2. When and why would you use a smart pointer over a bare pointer? Provide an example.
    - Bare pointers require cleaning up and safe deletion, you should not use bare pointers and instead use smart pointers which are safer.
    - You should always use a smart pointer. An example of this is a Dog that is a type of Animal. You would create the instance of a dog using make_shared/unique helper functions to create a smart pointer. You would then treat the smart pointer as the Dog.
3. What is the difference between a reference and a pointer?
    - A reference is an alias to a variable.
    - A reference cannot be instantiated without a reference to a variable. A pointer can be instantiated with no value (nullptr).
    - A pointer is simply a memory address, dereferencing it reads the value at the memory address.
4. What problem does the Type-Object design pattern help to solve?
    - Type-Object solves the problem of having many sub-classes of a base class. It instead replaces it by a member attribute to an “information” class that contains information related to that object instance.
    - Example: Plant is the base class and there are millions of plant types around the world. We don’t want to make a class for each type of plant, instead we’ll have a class called “Genetics” that is referenced by (contained as a member attribute) the plant class. This genetics class will contain information on the plant such as name, max height, weather conditions it likes and the best fertiliser to use.
    - We generally want to call a function on the genetics to create a new plant instance instead of making a new plant and pass it a genetics instance.
    - Genetics class defines a plant’s type. Each instance of a Genetics class is an object that represents a different type of plant.
5. List and briefly describe 2 variable storage methods available in C++17.
    - Stack – data is stored in the stack that is separate for each function. When a variable is declared locally in a function it is stored on the stack and is destroyed when the function ends.
    - Heap – data is stored in the heap will last the lifetime of the program or until deleted using the delete keyword. Data is stored here through pointers. The heap will grow and shrink during the lifetime of the program.

Output – 3 Questions

1. What is the output of the following code if a = 1 and b = 2?

```
auto lambda = [&]() {
  cout << "Hi" << endl;
  cout << a + b << endl;
  a = a + b;
};

lambda();
lambda();

cout << a << endl;
cout << b << endl;
```

```
Hi
3
Hi
5
5
2
```

2. What is the output of the following code?

```
int x[5] = {50, 100, 150, 200, 250};
cout << *x << endl;
++(*x);
*(x + 2) = 1000;
for (int i{0}; i < int(sizeof(x) / sizeof(x[0])); ++i)
{
    cout << x[i] << endl;
}
```

```
50
51
100
1000
200
250
```

3. What is the value of _a, _b, _c, _d, _e of the following code given the three class definitions below?

```
class Foo {
public:
    Foo() : _a{1}, _b{1}, _c{5} { }
    int Bar() {
        return _a + _b;
    }
protected:
    int _a;
    int _b;
    int _c;
};
class Bar {
public:
    Bar() : _d{3}, _e{2} { }
    int Foo() {
        return _d + _e;
    }
protected:
    int _d;
    int _e;
};

class FooBar : public Foo, public Bar {
public:
    FooBar() : Foo::Foo{}, Bar::Bar{} { }
    int Bar() {
        int b = _a;
        int a = _b;
        [=]() mutable {
          a = b;
        }();
        _a += a;
        _b += b;
        _e = Foo::Bar();
        _c = Bar::Foo();
        return 1;
    }
};
int main() {
    FooBar *barfoo = new FooBar();
    barfoo->Bar();
    delete barfoo;
}
```

```
_a = 2
_b = 2
_c = 7
_d = 3
_e = 4
```

Code – 2 Questions

1. Write a class structure that fits the scenario below. Methods should be declared and defined in the same class. There is no need to write a header file then a code file.

Animals all have nice collars. Collars are different depending on the animal type. Each collar has a bell of some sort that will return a string representing the way they ring. There are a few different types of animals: cats, dogs and cows. Cats have a Small Collar, dogs have a Big Collar and cows have a Giant Collar. Collars also have the name of the animal on them, animals do not have a name but do have a string description of the animal in general (you need to make this up). Animals can all be petted, if they are, they return various strings representing their reactions.

These answers were from the PASS session, I have edited them slightly to be more aligned to the question. There is more than one way to complete this question. The one in class did not use inheritance.

```
#include <iostream>
#include <memory>

class Collar {
public:
     Collar(const std::string &size, const std::string &name)
         : _size{size}, _name{name} {

     }

     const std::string size() {
```

```cpp
        return _size;
    }

    const std::string name() {
        return _name;
    }

private:
    std::string _size;
    std::string _name;
};

class Animal {
public:
    Animal(const std::shared_ptr<Collar> &collar, const std::string
&petReaction, const std::string &description)
        : _petReaction{petReaction}, _description{description},
_collar{collar} {

    }

    const std::string pet() {
        return _petReaction;
    }

    const std::string description() {
        return _description;
    }

    const std::shared_ptr<Collar> collar() {
        return _collar;
    }

private:
    std::string _petReaction;
    std::string _description;
    std::shared_ptr<Collar> _collar;
};

int main() {
    auto smallCollar = std::make_shared<Collar>("Small", "Doggo!");
    Animal dog = Animal{smallCollar, "Woof, woof!", "A small dog that is
nice to pet."};

    std::cout << dog.pet() << std::endl;
    std::cout << dog.collar()->name() << std::endl;

    return 0;
}
```

2. Using the decorator pattern for the scenario below you are to create the relevant classes and demonstrate after how you would create a Computer with 3 CPU's and 2 RAM and display the price. Once again, there is no need to create a separate file for declarations and definitions. Define in the same place as you declare.

Computers all have certain parts in them. We will focus on two of these parts: RAM and CPU. Forget how a computer is built and focus on these two parts. Computers can have various different amounts of RAM in GB and CPU core counts. We'll leave the concept of cores and GB out of this. What you need to know is that a computer can have any amount of both or none of one and some of the other. Using the decorator pattern, you need to make this possible.

All parts have a name and an attached price. Prices on all parts start from $10 for installation. RAM costs $50 each and CPU's cost $150 each. All computers have a name and a price.

The way to start this question is to draw a UML diagram then translate it to code. This is one way to do it, there may be other different variations on this way.

```
#include <memory>
#include <iostream>

using namespace std;

class Comp
{
public:
    Comp() : _name{"A computer"}, _price{0} {

    }
    virtual ~Comp() = default;
    virtual const string Name() const {
        return _name;
    }
    virtual int Price() const {
        return _price;
    }

protected:
    string _name;
    int _price{0};
};

class Part : public Comp
{
public:
    Part(const shared_ptr<Comp> &comp) : _comp{comp} {

    }
protected:
    shared_ptr<Comp> _comp;

};

class CPU : public Part
{
public:
    CPU(const shared_ptr<Comp> &comp) : Part{comp} {

    }
    int Price() const override {
```

```cpp
        return _comp->Price() + 160;
    }
    const string Name() const override {
        return _comp->Name() + " with a CPU";
    }
};

class RAM : public Part
{
public:
    RAM(const shared_ptr<Comp> &comp) : Part{comp} {

    }
    int Price() const override {
        return _comp->Price() + 60;
    }
    const string Name() const override {
        return _comp->Name() + " with a RAM";
    }
};

int main(){
    auto comp = make_shared<Comp>();
    auto compWithRam = make_shared<RAM>(comp);
    auto compWithRamWithCPU = make_shared<CPU>(compWithRam);

    cout << compWithRamWithCPU->Name() << endl;
    cout << compWithRamWithCPU->Price() << endl;

    return 0;
}
```