Programming in Modern C++: Assignment Week 6

Total Marks : 20

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology
Kharagpur – 721302
partha.p.das@gmail.com

August 24, 2023

Question 1

Consider the following program. *[MCQ, Marks 2]*

```
#include<iostream>
using namespace std;
class Class1{
public:
    void fun1() { cout << "C1" ; }
    virtual void fun2() { cout << "C2" ; }
};
class Class2 : public Class1{
public:
    void fun1() { cout << "C3" ; }
    void fun2() { cout << "C4" ; }
};
int main(){
    Class1 *t = new Class2();
    t->fun1();
    t->fun2();
    return 0;
}
```

What will be the output?

a) `C1C4`

b) `C2C4`

c) `C1C3`

d) `C2C3`

Answer: a)
Explanation:
As `fun1()` is a non-virtual function at the base class, for the `t->fun1()` function call, static binding is done. So, the function of the pointer type will be called.
As `fun2()` is a virtual function at the base class, for the `t->fun2()` function call, dynamic binding is done. So, the function of the object type will be called.

Question 2

Consider the code segment given below. *[MCQ, Marks 2]*

```
#include<iostream>
using namespace std;
class ClassA{
public:
    ClassA() { cout<<"1"; }
    ~ClassA() { cout<<"2"; }
};
class ClassB : public ClassA{
public:
    ClassB() { cout<<"3"; }
    virtual ~ClassB() { cout<<"4"; }
};
class ClassC : public ClassB{
public:
    ClassC() { cout<<"5"; }
    ~ClassC() { cout<<"6"; }
};
int main(){
    ClassA *t1 = new ClassC();
    delete t1;
    return 0;
}
```

What will be the output?

a) `135642`

b) `1352`

c) `13542`

d) `13562`

Answer: b)
Explanation:
When the object of class `classC` is created, it calls the constructor of class `classC`, which in turn calls the constructor of class `classB` and `classA` respectively. So, it will print `1 3 5`.
Whenever the object is deleted, it calls the destructor of class `classA` first. The destructor of class `classA` is not virtual, so it will not call the child class destructor. So, the final result will be `1352`.

Question 3

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>
using namespace std;
class student {
    string msg = "Hello ";
public:
    string greet(string name) { return (msg + name); }
};
class teacher : public student {
    string msg = "Hi ";
};
void print(student &ob, string name){ cout << ob.greet(name) << endl; } //LINE-1
int main() {
    student s1;
    teacher t1;
    print(s1, "Ravi"); //LINE-2
    print(t1, "Sir"); //LINE-3
    return 0;
}
```

What will be the output?

a)
```
Hello Ravi
Hi Sir
```

b)
```
Hello Ravi
Hello Sir
```

c)
```
Hi Ravi
Hello Sir
```

d)
```
Hi Ravi
Hi Sir
```

Answer: b)
Explanation:
At `LINE-2`, the function call print(s1, "Ravi"); invokes greet() from class `student`. Hence, the output is Hello Ravi.
At `LINE-3`, the function call `print(t1, "Sir");` is valid as it involves upper casting. However, as the base class is not virtual, it is a compile-time binding situation. Hence, at `LINE-1`, the function call `ob.greet(name)` invokes `greet()` from class `student`. Hence, the output is Hello Sir.

Question 4

Consider the code segment given below. *[MSQ, Marks 2]*

```
#include<iostream>
using namespace std;
class A{
public:
    virtual void f() = 0;
};
class B : public A{
    double data1;
public:
    void f(){ cout << "B "; }
};
class C : public B{
    double data2;
public:
    void f(){ cout << "C "; }
};

int main(){
    cout << sizeof(A) << " " << sizeof(B) << " " << sizeof(C);
    return 0;
}
```

What will be the output?

a) `0 8 16`

b) `8 16 24`

c) `4 12 20`

d) `8 16 16`

Answer: b)
Explanation:
Each pure virtual class maintains a pointer. Hence, sizeof(A) = 8.
Class B inherits class A and, additionally, has a data-member of type double. Hence, `sizeof(B) = sizeof(A) + sizeof(double) = 8 + 8 = 16`.
Class C inherits class B and additionally has a data-member of type double. Hence, `sizeof(C) = sizeof(B) + sizeof(double) = 16 + 8 = 24`.

Question 5

Consider the code segment. *[MSQ, Marks 2]*

```
#include<iostream>
using namespace std;
class A{
public:
    virtual void f(){ cout << "A::f() "; }
    void g(){ cout << "A::g() "; }
    void h(){ cout << "A::h() "; }
};
class B : public A{
public:
    void f(){ cout << "B::f() "; }
    void g(){ cout << "B::g() "; }
    void h(){ cout << "B::h() "; }
};
class C : public B{
public:
    void f(){ cout << "C::f() "; }
    void g(){ cout << "C::g() "; }
    virtual void h(){ cout << "C::h() "; }
};
int main(){
    C cb;
    B &bb = cb;
    bb.f();
    bb.g();
    bb.h();
    return 0;
}
```

What will be the output?

a) `A::f() B::g() C::h()`

b) `C::f() C::g() B::h()`

c) `C::f() B::g() B::h()`

d) `C::f() C::g() C::h()`

Answer: c)
Explanation:
In class B, the function f() is a virtual function. As bb refers to the object cb, the output will be `C::f() C::g() B::h()`.

Question 6

Consider the code segment given below. *[MSQ, Marks 2]*

```
#include<iostream>
using namespace std;
class classA{
public:
    virtual void f(){ cout << "A::f() "; }
};
class classB : public classA{
public:
    void f(){ cout << "B::f() "; }
};
class classC : public classB{
public:
    void f(){ cout << "C::f() "; }
};
int main(){
    classC *t = new classC;
    ________________; //LINE-1
    return 0;
}
```

Fill in the blank at `LINE-1` so that the program will print `A::f()`.

a) `t->f()`

b) `classA::t->f()`

c) `classA::f()`

d) `t->classA::f()`

Answer: d)
Explanation:
As t is a pointer to the object of class `classC`, we can call `f()` from class `classA` as `t->classA::f()` such that it will print `A::f()`.
Intentionally made as MSQ

Question 7

Consider the code segment below. *[MSQ, Marks 2]*

```
#include <iostream>
using namespace std;
class Test {
public:
    virtual void f() = 0;
};
void Test::f() { // Line 1
    cout << "Pure virtual function";
}
int main() {
    Test t; // Line 2
    Test *p = new Test(); // Line 3
    p->f(); // Line 4
    return 0;
}
```

The given program will not be compiled. Identify the correct reasons.

a) `Line-1: Pure virtual function in Test cannot have a body`

b) `Line-2: Cannot instantiate abstract class`

c) `Line-3: Invalid new expression for abstract class type`

d) `Line-4: Cannot de-reference a null pointer`

Answer: b), c)
Explanation: a) Pure virtual function can have a body. Incorrect reason.
b) Abstract base class (Test) cannot be instantiated. Correct reason.
c) We cannot use a new operator for an abstract base class. Correct reason.
d) Null pointer is checked at the run-time only. Incorrect reason.

Question 8

Consider the code segment given below. *[MCQ, Marks 2]*

```
#include<iostream>
using namespace std;
class Base{
public:
    virtual void fun() { }
};
class Derived : public Base{
public:
    void fun(double i) { }
};
int main(){
    Derived t1;
    Base *t2 = new Derived();
    t1.fun(); //Line-1
    t1.fun(3.14); //Line-2
    t2->fun(); //Line-3
    t2->fun(9.81); //Line-4
    return 0;
}
```

Which line/s will give you error?

a) `Line-1`

b) `Line-2`

c) `Line-3`

d) `Line-4`

Answer: a), d)
Explanation: The function `fun()` of class `Base` is overloaded in class `Derived`. So, the base class function becomes hidden for the derived class. So, `Line-1` will give an error. On the other hand, class `Base` doesn't have `fun(int)` in its definition. So, `Line-4` will give an error.

Question 9

Consider the code segment given below. *[MCQ, Marks 2]*

```
class Flower {
public:
    virtual void Petals() = 0 { cout << "Flower"; }
};
class FlowerWColor : public Flower {
    void Petals() { cout << "Flower with color"; }
};
class FlowerWOColor : public Flower {};
class Rose : public FlowerWColor {
public:
    void Petals() { cout << "Rose Flower"; }
};
class Jasmine : public FlowerWOColor {
public:
    void Petals() { cout << "Jasmine Flower"; }
};
class Sunflower : public FlowerWColor {
public:
    void Petals() { cout << "Sunflower flower"; }
};
```

Identify all abstract classes.

a) `Flower, FlowerWColor, FlowerWOColor`

b) `Flower, FlowerWOColor, Rose`

c) `Flower, FlowerWColor, FlowerWOColor, Sunflower`

d) `Flower`

Answer: b)
Explanation:
An abstract base class contains at least one pure virtual function. Moreover, a class derived from an abstract base class will also be abstract unless you override each pure virtual function in the derived class with non-pure ones. So, option b) is the correct answer.

Programming Questions

Question 1

Consider the following program. Fill in the blanks as per the instructions given below:

- Complete the destructor statement,
- Complete the constructor statement,

such that it will satisfy the given test cases. *Marks: 3*

```
#include<iostream>
using namespace std;
class B{
public:
    B(){ cout << "1 "; }
    B(double n){ cout << n << " "; }
    _____________; //LINE-1
};

class D : public B{
public:
    D(double n) : _________     //LINE-2
                { cout << n * 3 << " "; }
    D(){ cout << "3 "; }
    virtual ~D(){ cout << "4 "; }
};
B::~B(){ cout << "2 "; }
int main(){
    int i;
    cin >> i;
    B *pt = new D(i);
    delete pt;
    return 0;
}
```

Public 1

```
Input: 4
Output: 4 12 4 2
```

Public 2

```
Input: 2
Output: 2 6 4 2
```

Private

```
Input: 5
Output: 5 15 4 2
```

Answer:
Answer:
LINE-1: virtual ~B()

`LINE-2: B(n)`

Explanation:

At `LINE-1`, the destructor needs to be defined as a virtual destructor, so that if the derived class object gets deleted, it will be called automatically. Hence, `LINE-1` has to be filled with `virtual ~B();`. The initialization list required at `LINE-2` is `B(n)`.

Question 2

Consider the following program. Fill in the blanks as per the instructions given below.

- at `LINE-1`, define fun as pure abstract function,
- at `LINE-2`, complete constructor definition,
- at `LINE-3`, complete constructor definition,

such that it will satisfy the given test cases. *Marks: 3*

```
#include<iostream>
using namespace std;
class Test{
public:
    ___________________________ //Line-1
};
class ReTest1 : public Test{
    int d1;
public:
    ReTest1(int n) : ________{ } //Line-2
    void fun();
};
class ReTest2 : public Test{
    int d2;
public:
    ReTest2(int n) : _______{ } //Line-3
    void fun(){
        cout << d2 << " ";
    }
};
void ReTest1::fun(){
    cout << d1 << " ";
}
int main(){
    int i;
    cin>>i;
    Test *t1 = new ReTest1(i);
    Test *t2 = new ReTest2(i);
    t1->fun();
    t2->fun();
    return 0;
}
```

Public 1

```
Input: 2
Output: 4 6
```

Public 2

```
Input: 3
Output: 6 9
```

Private

```
Input: 4
Output: 8 12
```

Answer:

```
LINE-1:  virtual void fun() = 0;
LINE-2:  d1(2*n)
LINE-3:  d2(3*n)
```

Explanation:
We need to declare function `fun()` as pure virtual in the `Test` class so that we can call it using the `Test` class pointer. So, `Line-1` will be filled as `virtual void fun() = 0;`
`Line-2` and `Line-3` will be filled with `d1(2*n)` and `d2(3*n)` respectively in order to complete constructor definition.

Question 3

Consider the following program. Fill in the blanks as per the instructions given below.

- at `LINE-1`, declare the function show() as pure virtual function,
- at `LINE-2`, with appropriate function call,
- at `LINE-3`, with appropriate function header,

such that it will satisfy the given test cases. *Marks: 3*

```
#include <iostream>
using namespace std;
class shape{
protected:
    int a,b;
    shape(int x, int y) : a(x), b(y) {}
public:
    ______________________________    //LINE-1
};
class triangle : public shape{
public:
    triangle(int x, int y) : shape(x,y){}
    void Area();
    void show(){
        _______________; //LINE-2
        Area();
    }
};
__________________{ //LINE-3
    cout << a+b << " ";
}
void triangle::Area(){ cout << (0.5 * a * b); }
int main(){
    int a, b;
    cin >> a >> b;
    shape *sp = new triangle(a, b);
    sp->show();
    return 0;
}
```

Public 1

```
Input: 1
Output: 1, 6, 11
```

Public 2

```
Input: 5
Output: 5, 10, 15
```

Private

```
Input: 10
Output: 10, 15, 20
```

Answer:

```
LINE-1:  virtual void show() = 0;
LINE-2:  shape::show()
LINE-3:  void shape::show()
```

Explanation:
At `LINE-2`, the function `show()` from class shape must be called as `shape::show()`.
At `LINE-3`, the header for `show()` function must be `void shape::show()`.
The pure virtual function at `LINE-1` must be declared as `virtual void show() = 0;`.