Programming in Modern C++: Assignment Week 6

Total Marks : 25

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

February 21, 2025

Question 1

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

```
#include<iostream>
using namespace std;
class classA{
    public:
        void fun1() { cout << "1" ; }
        virtual void fun2() { cout << "3" ; }
};
class classB : public classA{
    public:
        void fun1() { cout << "2" ; }
        void fun2() { cout << "4" ; }
};
int main(){
    classA *t = new classB();
    t->fun1();
    t->fun2();
    return 0;
}
```

What will be the output?

a) `13`

b) `14`

c) `23`

d) `24`

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

Question 2

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

```
#include<iostream>
using namespace std;
class A{
    public:
        virtual void fun() { cout << "class1" << endl; }
};
class B : public A{
    public:
        void fun() { cout << "class2" << endl; }
};
int main(){
    B t1;
    A *t2 = new B();
    A *t3 = &t1;
    t2->fun();
    t3->fun();
    return 0;
}
```

What will be the output/error?

a)
```
class1
class1
```

b)
```
class1
class2
```

c)
```
class2
class1
```

d)
```
class2
class2
```

Answer: d)
Explanation:
The function `fun()` is declared as virtual in the base class. The derived class object is assigned to both pointer variables – `t2` and `t3` which are of type base. So, dynamic binding is done for both calls – `t2->fun()` and `t3->fun()`. So, option d) is correct.

Question 3

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

```
#include<iostream>
using namespace std;
int x = 0;
class Base{
    public:
        Base(){ x = x+3; }
        ~Base() { x = x-5; }
};
class Derived : public Base{
    public:
        Derived(){ x = x+7; }
        ~Derived(){ x = x-1; }
};
void fun(){
    Derived t;
    Base *t1 = new Derived();
    cout << x << " ";
    delete t1;
}
int main(){
    fun();
    cout << x;
    return 0;
}
```

What will be the output?

a) `20 8`

b) `20 9`

c) `17 7`

d) `17 6`

Answer: b)
Explanation:
When the function `fun` is called, an object of class `Derived` is created, which increase the value of global variable by $(7+3) = 10$. Again an object of class `Derived` is created with `new` which will increase global variable `x` by 10. So, 20 is printed first. When it is returned from function, destructor for both object would be called. In this process, for the object `(t1)`, only base class destructor is called because of non-virtual-ness. But, for the object `(t)`, both class destructor would be called. So, `x` is decreased by only 11. So, 9 will be printed.

Question 4

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

```
#include<iostream>
using namespace std;
class classA{
    public:
        classA() { cout<<"A "; }
        ~classA() { cout<<"~A "; }
};
class classB : public classA{
    public:
        classB() { cout<<"B "; }
        ~classB() { cout<<"~B "; }
};
class classC : public classB{
    public:
        classC() { cout<<"C "; }
        virtual ~classC() { cout<<"~C "; }
};
int main(){
    classA *t1 = new classC;
    delete t1;
    return 0;
}
```

What will be the output?

a) `A B C ~C ~B ~A`

b) `A B C ~C ~B`

c) `A B C ~B ~A`

d) `A B C ~A`

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

Question 5

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

```
#include<iostream>
using namespace std;
class base{
    int a;
    public:
        base(int i) : a(i) {}
        virtual void test(base *) { cout << a << endl; }
};
class derived : public base{
    int b;
    public:
        derived(int i=0, int j=0) : base(i), b(j) { }
        void test(derived *) { cout << b << endl; }
};
int main(){
    base *t1 = new derived(5,9);
    t1->test(new derived); //Line-1
    return 0;
}
```

What will be the output?

a) `0`

b) `5`

c) `9`

d) `garbage`

Answer: b)
Explanation:
The function in class `base` is overloaded in class `derived`. So, base class function is not available in derived class. So, the function call at `Line-1` will call base class function `base::fun(base *)`. This function call will print data member value of object of class `base` which is 5.

Question 6

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

```
#include<iostream>
using namespace std;
class A{
    public:
        void print(){ cout << "A "; }
};
class B : public A{
    public:
        ________________________________ //LINE-1
        void print(double d) { cout << "B "; }
};
int main(){
    B t1;
    t1.print(); //LINE-2
    t1.print(9.81);
    return 0;
}
```

Fill in the blank at `LINE-1` such that the output is `A B`.

a) `void print(){ cout << "A " ; }`

b) `using A::print;`

c) `void print(int i){ }`

d) `void print(){ }`

Answer: a), b)
Explanation:
In `LINE-2`, the object `t1` is trying to access the function `print()`, which can be inherited function from class base or overridden function in class derived. We can use the function of class base as it is and call it using the object `t1`. For that, `LINE-1` should be filled as `using A::print;`.
Otherwise, we can also override the same `print()` in class derived so that we can call using the object `t1`. For that, `LINE-1` should be filled as `void print(){ cout << "A "; }` such that we get the desired output.

Question 7

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

```
#include<iostream>
using namespace std;
class A{
    public:
        int a=5;
};
class B {
    public:
        int b=9;
};
class C {
    public:
        double c=3.14;
};
int main(){
    A *p;
    B t1;
    C t2;
    p = (A*)&t1;
    cout << p->a << " ";
    p = (A*)&t2;
    cout << p->a << " ";
    return 0;
}
```

What will be the output?

a) `5 5`

b) `9 3`

c) `9 3.14`

d) `9 <garbage-value>`

Answer: d)
Explanation:
Class `A` and `B` both have `int` data member and are unrelated class. Casting the object of class `B` will be rightly done and no information will be lost and prints 9. But in case of casting of class `C` object will hamper information due to type mismatch of data member. Hence, print garbage value.

Question 8

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

```
class Vehicle{
    public:
        virtual void run() = 0;
        virtual void stop() = 0;
};

class Car : public Vehicle{
};

class MotorCycle : public Vehicle{
    public:
        void run(){}
};

class Truck : public Car{
    public:
        void run(){}
        void stop(){}
};

class SportsCar : public Car{
    public:
        void run(){}
        virtual void runTurbo() = 0;
        void stop(){}
};

class SUV : public Car{
    public:
        void run(){}
        void stop(){}
};
```

Identify the abstract classes.

a) `Vehicle, Car, MotorCycle`

b) `Vehicle, Car, SUV`

c) `Vehicle, Car`

d) `Vehicle, Car, MotorCycle, SportsCar`

Answer: d)
Explanation:
A class having atleast one pure virtual function is a abstract class. At the same time, a pure virtual function remains pure virtual until it is overridden by the derived class. Hence, correct option is d).

Question 9

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

```
#include<iostream>
using namespace std;
class A{
    public:
        virtual void print(){ cout << "A "; }
};
class B : public A{
    public:
        void print(){ cout << "B "; }
};
class C : public B{
    public:
        void print(){ cout << "C "; }
};
int main(){
    A *cb = _________; //LINE-1
    cb->print();
    return 0;
}
```

Fill in the blank at `LINE-1` such that the program will print `B`.

a) `new A`

b) `new B`

c) `new C`

d) `It cannot be printed`

Answer: b)
Explanation:
If a function is declared as virtual, the function call depends on the type of object the pointer points to, not on the type of the pointer. Hence, to call function `print()` from `B` class, `LINE-1` need to be filled as `new B`.

Programming Questions

Question 1

Complete the program with the following instructions.

- Fill in the blank at `LINE-1` to complete the destructor declaration,
- Fill in the blanks at `LINE-2` to declare `fun()` as pure virtual function,
- Fill in the blank at `LINE-3` to complete the object initialization.

The program must satisfy the given test cases. *Marks: 3*

```
#include<iostream>
#include<cstring>
using namespace std;
class Base{
protected:
    string s;
public:
    Base(string c) : s(c){}
    _______ ~Base(){ } //LINE-1
    _________________________________; //LINE-2
};
class Derived : public Base{
public:
    Derived(string c) : Base(c) {}
    ~Derived();
    string fun(string x){
        return s+x;
    }
};
class Wrapper{
public:
    void fun(string a, string b){
        Base *t = ____________________; //LINE-3
        string i = t->fun(b);
        cout << i << " ";
        delete t;
    }
};
Derived::~Derived(){ cout << s << " "; }
int main(){
    string i, j;
    cin >> i >> j;
    Wrapper w;
    w.fun(i,j);
    return 0;
}
```

Public 1

```
Input: o k
Output: ok o
```

Public 2

```
Input: C ++
Output: C++ C
```

Private 1

```
Input: C #
Output: C# C
```

Answer:

```
LINE-1:  virtual
LINE-2:  virtual string fun(string) = 0
LINE-3:  new Derived(a)
```

Explanation:

The destructor of Base class needs to be declared as virtual in order to call Derived class destructor at the time of deletion. The function `fun()` can be declared as pure virtual function at `LINE-2` as `virtual string fun(string) = 0;`. We can't instantiate abstract class. So, `LINE-3` can be filled as `new Derived(a)`.

Question 2

Consider the following program with the following instructions.

- Fill in the blank at `LINE-1` to complete function declaration,
- Fill in the blank at `LINE-2` with appropriate statement so that global function `caller()` can access private member function `fun()` of class hierarchy.

The program must satisfy the sample input and output. *Marks: 3*

```
#include<iostream>
using namespace std;
class Base{
    int i;
    _______ void fun(); //LINE-1
public:
    Base(int x) : i(x) {}
    ______________________________; //LINE-2
};
class Derived : public Base{
    int j;
    void fun(){ cout << j; }
public:
    Derived(int x) : Base(x), j(10*x){}
};
void Base::fun(){ cout << i; }
void caller(Base &t){
    t.fun();
}
int main(){
    int x;
    cin >> x;
    Derived t(x);
    caller(t);
    return 0;
}
```

Public 1

```
Input: 8
Output: 80
```

Public 2

```
Input: 5
Output: 50
```

Private

```
Input: 50
Output: 500
```

Answer:

```
LINE-1:  virtual
```

`LINE-2: friend void caller(Base&)`

Explanation:

The function `fun()` at base class should be declared as virtual as we need to call derived class function `fun()` from the function `caller()` using base class type variable. Hence, `LINE-1` should be filled as virtual. The function `caller()` is accessing private member function of the class hierarchy. So, it should be friend of class `Base`. Hence, `LINE-2` should be filled as `friend void caller(Base&)`.

Question 3

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

- Fill in the blanks at `LINE-1 and LINE-2` with appropriate destructor declaration statements

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

```
#include<iostream>
using namespace std;
class Test{
protected:
    int n;
public:
    Test(int i) : n(i) { cout << ++n << " "; }
    ______________________; //Line-1
};

class DerivedTest : public Test{
public:
    ____________________; //Line-2
    DerivedTest(int i) : Test(2*i) { cout<< ++n << " "; }
};

Test::~Test() { cout<< --n << " "; }
DerivedTest::~DerivedTest() { cout<< --n << " "; }
int main(){
    int n;
    cin>>n;
    DerivedTest *d = new DerivedTest(n);
    Test *t = d;
    delete t;
    return 0;
}
```

Public 1

```
Input: 5
Output: 11 12 11 10
```

Public 2

```
Input: 2
Output: 5 6 5 4
```

Private

```
Input: 7
Output: 15 16 15 14
```

Answer:

```
LINE-1:  virtual ~Test()
LINE-2:  ~DerivedTest() OR virtual ~DerivedTest()
```

Explanation:
From the test cases, it can be noticed that both class destructor is being called while calling delete of pointer t of class type `Test`. So, we need to declare base class destructor as virtual. So, `Line-1` will be filled as `virtual ~Test();`. `Line-2` destructor declaration may or may not be virtual. So, it can be filled as `~DerivedTest(); OR virtual ~DerivedTest();`.