Programming in Modern C++: Assignment Week 5

Total Marks : 20

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

August 17, 2023

Question 1

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

```
#include <iostream>
using namespace std;
class ClassA {
    protected:
        int i;
    public:
        ClassA(int _i) : i(_i) {}
        void func() { cout << i << endl; }
};
class ClassB : public ClassA {
    public:
        ClassB(int _i) : ClassA(_i) {}
        void func(int x) { cout << i * x << endl; }
};
int main(){
    ClassB iA(5);
    iA.func(); //LINE-1
    return 0;
}
```

What will be the output/error?

a) `5`

b) `1`

c) `0`

d) `Compilation error at LINE-1`

Answer: d)
Explanation:
When we overload base class function in the derived class, the base class function will not be available to call using derived class object. So, it will be compilation error at `LINE-1`.

Question 2

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

```
#include <iostream>
using namespace std;
class base{
    protected:
        int t1;
};
class derived : public base{
    public:
        int t2;
        int sum(){ return t1 + t2; }
};
int main(){
    base b;
    derived d;
    b.t1 = 10;        //LINE-1
    d.t1 = 20;        //LINE-2
    d.t2 = 30;        //LINE-3
    cout << d.sum();  //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), b)
Explanation:
In public inheritance, protected members of base class, appear as protected members of derived class. Hence, in `LINE-2` the code d.t1 = 20; gives an error. Protected members cannot be accessed outside the class. So, `LINE-1` will give an error.

Question 3

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

```
#include<iostream>
using namespace std;
class Base {
    public:
        void f() { cout<< "Base class"; }
};
class Derived : public Base {
    public:
        void f() { cout<<"Derived class"; };
};
main() {
    Derived obj;
    _______________; //LINE-1
    return 0;
}
```

Fill in the blank at `LINE-1` so that the program will print `Base class`.

a) `Base.obj.f()`

b) `Base.obj::f()`

c) `obj.Base::f()`

d) `Base::obj.f()`

Answer: c)
Explanation:
As the function `f()` needs to be called from the base `class Base`, the appropriate syntax for the function call is `obj.Base::f()`.

Question 4

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

```
#include<iostream>
using namespace std;
class square{
    protected:
        int w;
    public:
        square(int _w) : w(_w){}
        int area(){ return w * w; }
};
class rectangle : public square{
    int h;
    public:
        rectangle(int _w, int _h) : ____________ { } //LINE-1
        int area(){ return w * h; }
};
int main(){
    rectangle *s = new rectangle(2,4);
    cout << s->area();
    return 0;
}
```

Fill in the blank at `LINE-1` so that the program will print `8`.

a) `square(_w), h(_h)`

b) `h(_h), square(_w)`

c) `w(_w), h(_h)`

d) `square(_w), square(_h)`

Answer: a), b)
Explanation:
The protected data member should be initialized from the derived class constructor. It can be done using option a) and b).

Question 5

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

```
#include <iostream>
using namespace std;
class Base {
    protected:
        int t1;
    public:
        Base(int _t1) : t1(_t1) { }
};
class Derived : ____________ { //LINE-1
    protected:
        int t2;
    public:
        Derived(int _t1, int _t2) : Base(_t1), t2(_t2) { }
};
class ReDerived : private Derived {
    public:
        ReDerived(int _t1, int _t2) : Derived(_t1, _t2) { }
        void print() { cout << t1 << " " << t2; }
};
int main() {
    ReDerived d(10, 20);
    d.print();
    return 0;
}
```

Fill in the blank at `LINE-1` so that the program will print `10 20`.

a) `private Base`

b) `protected Base`

c) `public Base`

d) `public ReDerived`

Answer: b), c)
Explanation:
As `ReDerived` class is already a child class of `Derived` class, and hence option (d) is wrong. If we use private inheritance (option (a)), then the data-member `t1` becomes private in `class Derived`. Hence, `t1` can not be inherited in `class ReDerived`. So, option (a) is also wrong. However, both options (b) and (c) are correct options.

Question 6

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

```
#include<iostream>
using namespace std;
class A{
    int x, y;
};
class B{
    protected:
        int z;
    public:
        void f(){ cout << "B::f()"; }
};
class C : public A, public B{
    A obj1;
};

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

What will be the output?

a) `0`

b) `12`

c) `16`

d) `20`

Answer: d)
Explanation:
As the `class C` inherits both the classes `A` and `B` and also has a data-member `A obj1;`, the `sizeof(C)` is calculated as follows:
`sizeof(C) = sizeof(A) + sizeof(B) + sizeof(obj1) = 8 + 4 + 8 = 20`

Question 7

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

```
#include<iostream>
using namespace std;
class Base{
public:
    void print() { cout << "Class Base" << endl; }
};
class Derived : private Base {
public:
    Derived() { _________________ } //LINE-1
};
int main(){
    Derived t1;
    return 0;
}
```

Fill in the blank at `LINE-1` so that the program will print `Class Base`.

a) `Base::print();`

b) `Base::print;`

c) `Base.print();`

d) `(new Base)->print();`

Answer: a), d)
Explanation: It can be seen that the `print()` function needs to be called from class `Derived` constructor in order to print `Class Base`. So, it can be called using the class name or temporary object. So, options a) and d) are correct.

Question 8

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

```
#include <iostream>
using namespace std;
class A {
    static int x1;
    int x2 = 5;
    public:
        void fun1() { cout << "f1" << endl; }
};
class B : public A {
    int d1 = 10;
};
int A::x1 = 0;
int main(){
    B t1;
    cout << sizeof(t1) << endl;
    return 0;
}
```

What will be the output?

a) 1

b) 4

c) 8

d) 12

Answer: c)
Explanation: static member doesn't take part in inheritance. So, derived class will not inherit base class data member `x1`. Only `x2` will be inherited from base class. So, the size of object `d` will be 8 bytes.

Question 9

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

```
#include <iostream>
using namespace std;
class B {
    public:
        void print() { cout << "B" << " "; }
};
class D : public B {
    public:
        void print() { cout << "D" << " "; }
};
int main(){
    B *a1 = new D();
    D *b1 = new D();
    a1->print();
    b1->print();
    return 0;
}
```

What will be the output?

a) `B D`

b) `D B`

c) `B B`

d) `D D`

Answer: a)
Explanation:
Invocation of a function depends on the type of the pointer. In our case, the first pointer if of type `B*`, and the second one is of type `A*`. Therefore, the output will be `B D`.

Programming Questions

Question 1

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

- Complete the inheritance statement at `LINE-1`,
- Complete the return statement at `LINE-2` and `LINE-3` to calculate the volume and surface area of a cube,

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

```
#include <iostream>
using namespace std;
class Volume {
    public:
        double getValue(int a) { return (a * a * a); }
};
class SurfaceArea {
    public:
        double getValue(int a) { return 6 * a * a; }
};
class Cube : ___________________________________ { //LINE-1
    int _a;
    public:
        Cube(int a) : _a(a) { }
        double getVolume() { return ___________________; } //LINE-2
        double getSurfaceArea() { return ___________________; } //LINE-3
};
int main() {
    int a;
    cin >> a;
    Cube c(a);
    cout << c.getVolume() << ", " << c.getSurfaceArea();
    return 0;
}
```

Public 1

```
Input: 4
Output: 64, 96
```

Public 2

```
Input: 3
Output: 27, 54
```

Private

```
Input: 5
Output: 125, 150
```

Answer:
Answer:

`LINE-1: public Volume, public SurfaceArea`
`LINE-2: Volume::getValue(_a)`
`LINE-3: SurfaceArea::getValue(_a)`

Explanation:

The class Cube must inherit from both Volume and SurfaceArea classes. So at `LINE-1`, we use `class Cube : public Volume, public SurfaceArea`

Note that any of public, protected, or private inheritance will work in this case, classes may be put in any order, and private inheritance may be implied by skipping the specifier/s for inheritance. Hence, there are several fill-ups that will work as long as both classes `Volume` and `SurfaceArea` are listed. The function `getValue()` is defined in both `Volume` and `SurfaceArea` classes. To resolve the ambiguity, we need to use `Volume::getValue(_a)` at `LINE-2` to call `getValue()` from class Volume and `SurfaceArea::getValue(_a)` to call `getValue()` from class `SurfaceArea`.

Question 2

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

- at `LINE-1` with appropriate keyword,
- at `LINE-2` and `LINE-3` with appropriate constructor statements

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

```
#include <iostream>
using namespace std;
class Vehicle{
    string vehicleName;
    int noOfWheels;
    protected:
        Vehicle(string s, int w) : vehicleName(s), noOfWheels(w) { }
    public:
        ________ void vehicleDetails(const Vehicle&); //LINE-1
};
class Twowheeler : public Vehicle{
    public:
        Twowheeler(string n) : ____________ { } //LINE-2
};
class Fourwheeler : public Vehicle{
    public:
        Fourwheeler(string n) : ____________ { } //Line-3
};
void vehicleDetails(const Vehicle &v){
    cout << v.vehicleName << ": ";
    if(v.noOfWheels == 2)
        cout << "Two Wheeler";
    else if(v.noOfWheels == 4)
        cout << "Four Wheeler";
}
int main(){
    string s;
    int n;
    Vehicle *v;
    cin >> s >> n;
    if(n==2)
        v = new Twowheeler(s);
    else if(n==4)
        v = new Fourwheeler(s);
    vehicleDetails(*v);
    return 0;
}
```

Public 1

```
Input: Bus 4
Output: Bus: Four Wheeler
```

Public 2

```
Input: Bike 2
Output: Bike: Two Wheeler
```

Private

```
Input: Truck 4
Output: Truck: Four Wheeler
```

Answer:

```
LINE-1:  friend
LINE-2:  Vehicle(n,2)
LINE-3:  Vehicle(n,4)
```

Explanation:

The global function `vehicleDetails` needs access private members of class Vehicle. So, it should be a friend function of class Vehicle. `LINE-1` will be filled with friend.

From `LINE-2`, constructor of class Vehicle needs to be called with noOfWheels value as 2. It can be done as `Vehicle(n, 2)`.

In the similar way, `LINE-3` will be filled as `Vehicle(n, 4)`.

Question 3

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

- at `LINE-1` with appropriate inheritance statement,
- at `LINE-2` with appropriate constructor statement

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

```
#include<iostream>
using namespace std;
class B1{
    protected:
        int b1;
    public:
        B1(int b) : b1(b){}
};
class B2{
    protected:
        int b2;
    public:
        B2(int b) : b2(b){}
};
class D : ____________________________{ //LINE-1
    int d;
public:
    D(int x) : _________________________________{} //LINE-2
    void show(){
        cout << d << ", " << b1 << ", " << b2;
    }
};
int main(){
    int x;
    cin >> x;
    D t1(x);
    t1.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: public B1, public B2`

`LINE-2: B1(x+5), B2(x+10), d(x)`

Explanation:

The function `show()` of class D is accessing protected member of both class B1 and class B2. This can be done when D class is inherited from class B1 and B2. So, `LINE-1` will be filled as `public B1, public B2` or protected inheritance in any order.

As per the test cases, the constructor at `LINE-2` needs to be filled as `B1(x+1), B2(x+2), d(x)` or in any order.