Programming in C++: Assignment Week 5

Total Marks : 27

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

September 18, 2020

Question 1

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

```
#include <iostream>
using namespace std;

class base {
    static int x1;
    int x2 = 5;
public:
    void f1() { cout << "f1" << endl; }
};

class derived : public base {
    int d1 = 10;
};

int base::x1 = 0;

int main() {
    derived d;

    cout << sizeof(d) << endl;

    return 0;
}
```

What will be the output of the above code (consider `sizeof(int) = 4`)?

a) 12

b) 8

c) 4

d) 1

Answer: b)
Explanation:
static members are not part of an object layout, and hence play no role 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 2

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

```
#include <iostream>
using namespace std;

class base {
public:
    void f1() { cout << "base.f1" << endl; }
};

class derived : public base {
public:
    void f1(int a) { cout << "derived.f1" << endl; }
};

int main() {
    derived d;

    d.f1();                // LINE-1

    return 0;
}
```

What will be the output/error?

a) `base.f1`

b) `derived.f1`

c) `base.f1`
   `derived.f1`

d) `Compilation error at LINE-1:  no matching function for call derived.f1()`

Answer: d)
Explanation:
Here we have actually overloaded base class function `void base::f1()` in the derived class as `void derived::f1(int)`. So the base class function `void base::f1()` will not be available to call in `LINE-1` using derived class object. So, it will be compilation error at `Line-1`.

Note that if we had `void derived::f1()`, then it would have worked as we would have a valid overriding.

Question 3

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

```
#include <iostream>
using namespace std;

class A {
public:
    void print() { cout << "Class A" << endl; }
};

class B : public A {
public:
    void print() { cout << "Class B" << endl; }
};

int main() {
    A *a1 = new A();
    A *b1 = new B();

    a1->print();
    b1->print();

    return 0;
}
```

What will be the output?

a)
```
Class A
Class B
```

b)
```
Class A
Class A
```

c)
```
Class B
Class A
```

d)
```
Class B
Class B
```

Answer: b)
Explanation:
Binding of a member function in a call depends on the type of the pointer and whether or not member function is virtual or not. In our case, both pointers are having `A` class (base) type and only static binding is used. So, both pointer will call base class function `A::print(.)`.

Question 4

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

```
#include <iostream>
#include <string>
using namespace std;

class A {
    string s1 = "Hello";
public:
    string get_str() { return s1; }
};

class B : public A {
    string s2 = "Hi";
};

void print(A &a) {
    cout << a.get_str() << endl;
}

int main() {
    A t1;
    B t2;

    print(t1);      // LINE-1
    print(t2);        // LINE-2

    return 0;
}
```

What will be the output/error?

a) `Hello`
   `Hello`

b) `Hello`
   `Hi`

c) `Hi`
   `Hello`

d) `Compilation error at LINE-1:  argument mismatch.`

Answer: a)
Explanation:
The global function `print(A&)` called at `LINE-1`, results in printing `Hello`.
While calling the same function with argument as an object of class `B` (at `LINE-2`), it will not give any compilation error. Because, an implicit up-casting will happen from derived class to base class. The function call at `LINE-2` will consider only the base class portion of the object `t2`, taken as argument. As a result, in both function call it will print as `Hello`. So, option (a) is correct.

Question 5

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

```
#include <iostream>
using namespace std;

class myClassA {
public:
    int a;
    myClassA(int x) : a(x) { }
};

class myClassB : private myClassA {
    int b;
public:
    myClassB(int x, int y) : b(y), myClassA(x) { }
};

int main() {
    myClassB t1(1, 2);
    myClassA t2(5);

    cout << t1.a;        // LINE-1
    cout << t2.a;        // LINE-2

    return 0;
}
```

Which line will give compilation error in the `main()` function?

a) `LINE-1`

b) `LINE-2`

c) `Both LINE-1 and LINE-2`

d) `No Compilation Error`

Answer: a)
Explanation:
The data member `a` is declared as `public` in class `myClassA`. So, we can access this from anywhere using the `myClassA` object. So, `LINE-2` is fine. But same data member becomes private when we inherit `myClassA` from class `myClassB` as it is private inheritance. So, `LINE-1` will give compilation error.

Question 6

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

```
#include <iostream>
using namespace std;

class A {
public:
    A() { cout << "A "; }
    ~A() { cout << "~A "; }
};

class B : public A {
public:
    B() { cout << "B "; }
    ~B() { cout << "~B "; }
};

class C : public A {
    B b;
public:
    C() { cout << "C "; }
    ~C() { cout << "~C "; }
};

int main() {
    C t1;

    return 0;
}
```

What will be the output?

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

b) `A C ~C ~A`

c) `A A B C ~C ~B ~A ~A`

d) `A A B C ~A ~A ~B ~C`

Answer: c)
Explanation: When object of class `C` is being instantiated, constructor of `class A` is called which will print "A" first. Then data member of class `C` is created, which will again print "A" then print "B". Then at last "C" is printed from constructor of class `C`. After the end of `main()` function, reverse of already printed sequence will be printed from the destructor of the classes. So, the answer is (c).

Question 7

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

```
#include <iostream>
using namespace std;

class A {
public:
    void print() { cout << "Function print" << endl; }
};

class B : private A {
public:
    B() { _____________ }   // LINE-1
};

int main() {
    B t1;

    return 0;
}
```

Fill in the blank at `LINE-1` so that it will print: `Function print`.

a) `print();`

b) `A::print;`

c) `A.print();`

d) `A::print();`

Answer: a), d)
Explanation:
It can be seen that the `print()` function needs to be called from class `B` constructor in order to print "Function print". So, option a) and d) are correct.

Question 8

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

```
#include <iostream>
using namespace std;

class A {
public:
    A(int i) { cout << "A::" << i << " "; }
    ~A() { cout << "~A "; }
};

class B : public A {
public:
    B(int i) : A(i++) { cout << "B::" << i << " "; }
    ~B() { cout << "~B "; }
};

class C : public B {
public:
    C(int i) : B(i++) { cout << "C::" << i << " "; }
    ~C() { cout << "~C "; }
};

void f() {
    static C c(0);
}

int main() {
    f();

    C c(5);

    return 0;
}
```

What will be the output?

a) `A::0 B::1 C::1 A::5 B::6 C::6 ~C ~B ~A ~C ~B ~A`

b) `A::0 B::1 C::2 ~C ~B ~A A::5 B::6 C::7 ~C ~B ~A`

c) `A::0 B::1 C::1 ~C ~B ~A A::5 B::6 C::6 ~C ~B ~A`

d) `A::0 B::1 C::2 A::5 B::6 C::7 ~C ~B ~A ~C ~B ~A`

Answer: a)
Explanation: When we instantiate an object of a derived class its base classes will be also instantiated in top-down order in class hierarchy. The destructors will be invoked in exactly reverse order.
We first create an object of derived class as `static C c(0);`. Hence the constructors will be invoked in top-down order and the output is `A::0 B::1 C::1`. As the scope of `c` is within the function `f()` but it is declared as static, destructors for this object will be invoked at the end of main function.

Next object is created as `C c(5);` whose scope is local within `main()`. Hence the output for instantiation of object is `A::5 B::6 C::6` and after the scope the destructors will be invoked with the output: ∼`C` ∼`B` ∼`A`.
Finally, the function static object will be destructed after `main()` function. Hence the output will be ∼`C` ∼`B` ∼`A`
So, a) is correct.

Question 9

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

```
#include <iostream>
using namespace std;

class Base {
protected:
    int X;
public:
    Base(int i = 0) : X(i) { }
};

class Derived : public Base {
    Base b;
public:
    Derived(Base b1, int i = 0) : Base(i), b(b1) { }
    void print1() { cout << X << endl; }         // LINE-1
    void print2() { cout << b.X << endl; }       // LINE-2
};

int main() {
    Base b(5);
    Derived d(b, 10);

    d.print1();
    d.print2();

    return 0;
}
```

What will be the output/error?

a) `10 5`

b) `5 10`

c) `Compilation error at LINE-1`

d) `Compilation error at LINE-2`

Answer: d)
Explanation:
In the both `print` function, they are trying to access protected data member `X` of class Base. `Derived` class has two data members. One is `b` of type `Base` which is `private` and the other is `X` which is coming from `Base` class due to inheritance property and it is `protected`. So in `Derived` class, variable `X` is easily accessible from public function `print1()` in `LINE-1` but in `LINE-2`, `print2()` function is trying to access data member of object `b` which is `protected` in `Base` class. So it can't be accessed from outside of `Base` class.

Programming Questions

Question 1

Consider the following program. Fill in the blank at `LINE-1` for appropriate inheritance. Fill in the blanks at `LINE-2` and `LINE-3` to complete the function definition such that it can satisfy the given test cases. *Marks: 3*

```
#include <iostream>
using namespace std;

#define PI 3.14

class Volume {
public:
    double getValue(int r) { return (4 * PI * r * r * r / 3); }
};

class SurfaceArea {
public:
    double getValue(int r) { return 4 * PI * r * r; }
};

class Sphere : ____________________ {              // LINE-1
    int _r;
public:
    Sphere(int a) : _r(a) { }
    double getVolume() { return __________; }      // LINE-2
    double getSurfaceArea() { return __________; } // LINE-3
};

int main() {
    int a;
    cin >> a;

    Sphere s(a);
    cout << s.getVolume() << ", " << s.getSurfaceArea();

    return 0;
}
```

Public 1

```
Input:  2
Output:  33.4933, 50.24
```

Public 2

```
Input:  5
Output:  523.333, 314
```

Private

```
Input:  10
Output:  4186.67, 1256
```

Answer:

`LINE-1: public Volume, public SurfaceArea` OR any variant of multiple inheritance
`LINE-2: Volume::getValue(_r)`
`LINE-3: SurfaceArea::getValue(_r)`

Explanation:

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

```
class Sphere :  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(_r)` at LINE-2 to call `getValue()` from class `Volume` and `SurfaceArea::getValue(_r)` to call `getValue()` from class `SurfaceArea`.

Question 2

Consider the following program. Fill in the blanks in `LINE-1` such that global function `dist` can access private member of class `Coordinate`. Fill in the blanks at `LINE-2` and `LINE-3` to complete constructor definition. Consider the given test cases. *Marks: 3*

```
#include <iostream>
#include <cmath>
using namespace std;

class Coordinate {
    int x, y, z;
public:
    Coordinate(int _x, int _y, int _z = 0) :
        x(_x), y(_y), z(_z) {    }
    __________ double dist(Coordinate &c1, Coordinate &c2); // LINE-1
};

class TwoDCoordinate : public Coordinate {
public:
    TwoDCoordinate(_____________________) : // LINE-2
        Coordinate(_x, _y, _z) { }
};

class ThreeDCoordinate : public Coordinate {
public:
    ThreeDCoordinate(_____________________) : // LINE-3
        Coordinate(_x, _y, _z) { }
};

double dist(Coordinate &c1, Coordinate &c2) {
    return sqrt(pow((c1.x - c2.x), 2) +
                pow((c1.y - c2.y), 2) +
                pow((c1.z - c2.z), 2));
}

int main() {
    int x1, y1, x2, y2, z2;
    cin >> x1 >> y1 >> x2 >> y2 >> z2;

    TwoDCoordinate t1(x1, y1);
    ThreeDCoordinate t2(x2, y2, z2);

    cout << dist(t1, t2) << endl;

    return 0;
}
```

Public 1

```
Input:  1 2 3 4 5
Output:  5.74456
```

Public 2

```
Input:  1 1 2 2 1
Output:  1.73205
```

Private

```
Input:  1 0 2 1 1
Output:  1.73205
```

Answer:
LINE-1: `friend`
LINE-2: `int _x, int _y, int _z=0`
LINE-3: `int _x, int _y, int _z`
Explanation:
The global function `dist(.)` should be a friend of class `Coordinate` in order to access private data members of the class. So, `Line-1` will be filled as `friend`. We need to mention default value for `z` coordinate as 0 in `TwoDCoordinate` class constructor definition. So, `Line-2` will be filled as `int _x, int _y, int _z=0`. Constructor at `Line-3` will be completed as `int _x, int _y, int _z`.

Question 3

Consider the following program. Fill in the blank at `LINE-1` such that global function `vehicleDetails` can access private member of class `Vehicle`. Fill in the blank at `LINE-2` and `LINE-3` to call constructor of `Vehicle` class. After all fill ups, the program must satisfy the given test cases.

Marks: 3

```
#include <iostream>
#include <string>
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 Threewheeler : public Vehicle { public:
    Threewheeler(string n) : __________ { }         // Line-3
};

void vehicleDetails(const Vehicle &v) {
    cout << v.vehicleName << ": ";
    if (v.noOfWheels == 2)
        cout << "Two Wheeler";
    else
        cout << "Three Wheeler";
}

int main() {
    string s;
    int n;
    Vehicle *v;

    cin >> s >> n;

    if (n == 2)
        v = new Twowheeler(s);
    else
        v = new Threewheeler(s);

    vehicleDetails(*v);

    return 0;
}
```

Public 1

```
Input:  Auto 3
Output:  Auto:  Three Wheeler
```

Public 2

```
Input:  Bicycle 2
Output:  Bicycle:  Two Wheeler
```

Private

```
Input:  Cycle 2
Output:  Cycle:  Two Wheeler
```

Answer:

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

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, 3)`.