Programming in C++: Assignment Week 6

Total Marks : 30

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 Mark 2]*

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

class A {
public:
    void fun1() { cout << "A::fun1" << endl; }
    virtual void fun2() { cout << "A::fun2" << endl; }
};

class B : public A {
public:
    void fun1() { cout << "B::fun1" << endl; }
    void fun2() { cout << "B::fun2" << endl; }
};

int main() {
    A *t = new B();

    t->fun1();
    t->fun2();

    return 0;
}
```

What will be the output?

a)
```
A::fun1
B::fun2
```

b)
```
A::fun1
A::fun2
```

c)
```
B::fun1
B::fun2
```

d) `B::fun1`
`A::fun2`

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

Question 2

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

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

class Myclass {
public:
    virtual void fun() = 0;
};

void Myclass::fun() {                   // LINE-1
    cout << "Pure virtual function";
}

int main() {
    Myclass m;                          // LINE-2
    Myclass *p = new Myclass();         // LINE-3

    p->fun();                           // LINE-4

    return 0;
}
```

The given program does not compile. Identify the correct reason/s.

a) LINE-1: Pure virtual function in `Base` cannot have a body

b) LINE-2: Cannot instantiate abstract class

c) LINE-3: Invalid operator 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 (`Base`) cannot be instantiated. Correct reason

c) We cannot use new operator for an abstract base class. Correct reason

d) Null pointer is checked at the run-time only. Incorrect reason

Question 3

What will be the output of the following program?. *[MCQ, Marks 2]*

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

class base {
public:
    virtual void fun() { cout << "base::fun" << endl; }
};

class derived : public base {
public:
    void fun() { cout << "derived::fun" << endl; }
};

int main() {
    derived t1;
    base *t2 = new derived();
    base *t3 = &t1;

    t2->fun();
    t3->fun();

    return 0;
}
```

a) `base::fun`
   `base::fun`

b) `base::fun`
   `derived::fun`

c) `derived::fun`
   `derived::fun`

d) `derived::fun`
   `base::fun`

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

Question 4

What will be the output of the below program? *[MCQ, Marks 2]*

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

int x = 0;

class myClass {
public:
    myClass() { x++; }
    ~myClass() { x--; }
};

class test : public myClass {
public:
    test() { x += 5; }
    ~test() { x -= 2; }
};

void fun() {
    test t;
    myClass *t1 = new test();

    cout << x << " ";

    delete t1;
}

int main() {
    fun();

    cout << x;

    return 0;
}
```

a) 12 8

b) 12 6

c) 10 8

d) 10 6

Answer: a)
Explanation: When the function `fun` is called, a static object of class `test` is created, which increase the value of global variable by (5+1)=6. Again an object of class `test` is created which will increase global variable `x` by 6. So, 12 is printed first. When it is returned from function, destructor for both object is called. In this process, for the dynamic object (`t1`), only base class destructor is called because of not being virtual. But, for the static object (`t`), both class destructor will be called. So, `x` is decreased by only 4. So, 8 will be printed.

Question 5

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

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

class X {
public:
    virtual void fun() { }
};

class Y : public X {
public:
    void fun(int i) { }
};

int main() {
    Y t1;
    X *t2 = new Y();

    t1.fun();        // LINE-1
    t1.fun(3);       // LINE-2
    t2->fun();       // LINE-3
    t2->fun(3);      // LINE-4

    return 0;
}
```

Which line/lines 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 `X` is overloaded in class `Y`. So, base class function become hidden for derived class. So, `LINE-1` will give error. On the other hand, class `X` doesn't have `fun(int)` in its definition. So, `LINE-4` will give error.

Question 6

Consider the program below. *[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 "; }
    virtual ~B() { cout << "~B "; }
};

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

int main() {
    A *t1 = new C;

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

This leads to a partial wrap-up of the object `C` and a corresponding system exception as well.

Question 7

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

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

class A {
public:
    virtual void f1() { cout << "A::f1" << endl; }
    void f2() { cout << "A::f2" << endl; }
};

class B : public A {
public:
    void f1() { cout << "B::f1" << endl; }
    virtual void f2() { cout << "B::f2" << endl; }
};

class C : public B {
public:
    void f1() { cout << "C::f1" << endl; }
    void f2() { cout << "C::f2" << endl; }
};

int main() {
    A *a = new C();

    a->f1();
    a->f2();

    return 0;
}
```

What will be the output of the above code.

a) `A::f1`
   `B::f2`

b) `B::f1`
   `C::f2`

c) `A::f1`
   `C::f2`

d) `C::f1`
   `A::f2`

Answer: d)
Explanation:
Whenever we are declaring a function as `virtual` in a class, it will remain virtual throughout the classes inherited from that class. Both functions are called using pointer of class `A` which is assigned by an object of class `C`. So, in this case, the function `f1` is called from class `C` because of dynamic binding and function `f2` is called from class `A` because of static binding property.

Question 8

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

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

class A {
    int a;
public:
    A(int i) : a(i) { }
    virtual void fun(A *) { cout << a << endl; }
};

class B : public A {
    int b;
public:
    B(int i = 0, int j = 0) : A(i), b(j) {    }
    void fun(B *) { cout << b << endl; }
};

int main() {
    A *t1 = new B(1, 2);

    t1->fun(new B);        // LINE-1

    return 0;
}
```

What will be the output?

a) `0`

b) `1`

c) `2`

d) `garbage`

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

Question 9

Identify the abstract class/es from the following code snippet. *[MCQ, Marks 2]*

```
class Flower {
public:
    virtual void Petals() = 0 { cout << "Flower"; }
};

class FlowerWSmell : public Flower {
    void Petals() { cout << "Flower with smell"; }
};

class FlowerWOSmell : public Flower { };

class Rose : public FlowerWSmell {
public:
    void Petals() { cout << "Rose Flower"; }
};

class Jasmine : public FlowerWSmell {
public:
    void Petals() { cout << "Jasmine Flower"; }
};

class Sunflower : public FlowerWOSmell {
public:
    void Petals() { cout << "Sunflower flower"; }
};

class Hibiscus : public FlowerWOSmell { };
```

a) `Flower, FlowerWSmell, FlowerWOSmell`

b) `Flower, FlowerWOSmell, Hibiscus`

c) `Flower, FlowerWSmell, FlowerWOSmell, 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 Assignments

Question 1

Consider the program below. Fill in the blank at `LINE-1` with abstract function definition for `fun()`, and at `LINE-2` and `LINE-3` fill in the blanks with appropriate initialization list so that it satisfies the given test cases. *Do not change any other part of the code.* *[Marks 3]*

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

class Base {
public:
    ____________________; // LINE-1: Define fun as pure virtual function
};

class Derived1 : public Base {
    int d1;
public:
    Derived1(int n) : __________ { } // LINE-2: Complete constructor definition
    void fun() {
        cout << d1 << " ";
    }
};

class Derived2 : public Base {
    int d2;
public:
    Derived2(int n) : __________ { } // LINE-3: Complete constructor definition
    void fun() {
        cout << d2 << " ";
    }
};

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

    Base *b1 = new Derived1(i);
    Base *b2 = new Derived2(i);

    b1->fun();
    b2->fun();

    return 0;
}
```

Public Test Case 1

```
Input: 5
Output: 5 10
```

Public Test Case 2

```
Input: 10
Output: 10 20
```

Private Test Case

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

Answer:

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

Explanation:

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

Question 2

Consider the program below. Fill in the blank at `LINE-1` with abstract function declaration for `Salary()`. Fill in the at `LINE-2` with proper header of the function. Fill in the blanks at `LINE-3` and `LINE-4` with appropriate statement to call function `computeAllowance()` such that that it satisfies the given test cases. *Do not change any other part of the code. [Marks 3]*

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

class Professor { double allowance = 10;
public:
    ____________________;         // LINE-1
    double computeAllowance(int);
};

____________________(int basic) {    // LINE-2
    return (basic*allowance / 100);
}

class HOD : public Professor { int basic;
public:
    HOD(int _b) : basic(_b) {    }
    void Salary() {
        double a = ____________________;    // LINE-3: Call computeAllowance()
        cout << "HOD Salary = " << (basic + a) << endl;
    }
};

class Director : public Professor { int basic;
public:
    Director(int _b) : basic(_b) {    }
    void Salary() {
        double a = ____________________;    // LINE-4: : Call computeAllowance()
        cout << "Director Salary = " << (basic + a) << endl;
    }
};

int main() {
    int h, d;
    Professor *p;

    cin >> h >> d;

    p = new HOD(h);
    p->Salary();

    p = new Director(d);
    p->Salary();

    return 0;
}
```

Public Test Case 1

```
Input: 500 1000
Output:
HOD Salary = 550
Director Salary = 1100
```

Public Test Case 2

```
Input: 1000 1900
Output:
HOD Salary = 1100
Director Salary = 2090
```

Private Test Case

```
Input: 50000 80000
Output:
HOD Salary = 55000
Director Salary = 88000
```

Answer:

```
LINE-1:  virtual void Salary() = 0;
LINE-2:  double Professor::computeAllowance
LINE-3 & Line-4:  Professor::computeAllowance(basic)
```

Explanation:

We need to declare function `Salary(.)` as pure virtual in `LINE-1` so that the function can be defined differently in derived classes. So, it can be filled as

```
virtual void Salary() = 0;
```

At `LINE-2`, we need to complete function header with scope resolution operator as it is being defined outside of the class. So, it can be filled as

```
double Professor::computeAllowance(int basic)
```

At `LINE-3` and `LINE-4`, we need to call base class `computeAllowance` function in order to compute allowance amount. So, it can be filled as

```
double a = Professor::computeAllowance(basic)
```

Question 3

Consider the program below. Fill in the blanks at `LINE-1` and `LINE-2` to define appropriate destructors such that it matches given test cases. *Do not change any other part of the code. Marks: 3*

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

class Test {
public:
    Test() { cout << "1 "; }
    __________;    // LINE-1
};

Test::~Test() { cout << "2 "; }

class DerivedTest : public Test {
public:
    DerivedTest() { cout << "3 "; }
    DerivedTest(int i) { cout << 2 * i << " "; }
    __________;    // LINE-2
};

DerivedTest::~DerivedTest() { cout << "4 "; }

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

    DerivedTest *d = new DerivedTest(n);
    Test *t = d;

    delete t;

    return 0;
}
```

Public Test Case 1

```
Input: 3
Output: 1 6 4 2
```

Public Test Case 2

```
Input: 5
Output: 1 10 4 2
```

Private Test Case

```
Input: 2
Output: 1 4 4 2
```

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();`

Question 4

Fill in the blanks at `LINE-1` with proper access modifier, at `LINE-2` so that global function `addition` can access private data member of `Base` class and at `LINE-3` to call `Base` class `show()` function such that it matches the given test cases. *Do not change any other part of the code.*

Marks: 3

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

class Base {
    int b;
public:
    Base(int n) : b(n) { }
    __________ void show() {        // LINE-1
        cout << b << " ";
    }
    __________;        // LINE-2
};

class Derived : public Base {
    int d;
public:
    Derived(int m, int n) : Base(m), d(n) { }
    void show() {
        __________;            // LINE-3
        cout << d << " ";
    }
};

void addition(Base &x, Base &y) {
    x.b = x.b + y.b;
}

int main() {
    int m, n;
    cin >> m >> n;

    Base *t1 = new Derived(m, n);
    Base *t2 = new Base(n);

    addition(*t1, *t2);
    t1->show();

    return 0;
}
```

Public Test Case 1

```
Input: 5 6
Output: 11 6
```

Public Test Case 2

```
Input: 2 5
Output: 7 5
```

Private Test Case

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

Answer

```
LINE-1:  virtual
LINE-2:  friend void addition(Base&, Base&)
LINE-3:  Base::show()
```

Explanation:

`show()` function in `Base` class should be declared as `virtual` such that `Derived` class function can be called using base class pointer type. Global function `addition` can access private data members of `Base` class only if it is a friend function of that class. So, `LINE-2` can be filled as `friend void addition(Base &, Base &)`. `Base` class `show` function can be called at `LINE-3` as, `Base::show()`.