Programming in C++: Assignment Week 3

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, 2020

Question 1

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

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

class Sample {
    string name;
public:
    Sample() {
        cout << "s" << " ";
    }
    Sample(string s) : name(s) {
        cout << name << " ";
    }
};

int main() {
    Sample s1;    // LINE-1
    Sample *s2 = new Sample("s2");
    Sample *s3;

    new Sample("s4");

    return 0;
}
```

What will be the output?

a) `compilation error:  at LINE-1`

b) `s s2 s s4`

c) `s2 s s4`

d) `s s2 s4`

Answer: d)
Explanation:
The statement `Sample s1();` is not an error, but it does not instantiate an object. Hence option a) can be eliminated. Note that `Sample s1;` would actually instantiate an object and print an `"s"`.
Statement `Sample *s2 = new Sample("s2");`, instantiate an object, and call parametrized constructor. Hence it prints `s2`.
Statement `Sample *s3;`, just create a pointer and do not instantiate an object.
Statement `new Sample("s4");`, creates a temporary object, and call the parametrized constructor. Hence prints `s4`.
Hence, the correct option is d).

Question 2

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

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

int i = 0;

class myClass {
public:
    myClass() { i = 1; }
    ~myClass() { i = 5; }
};

void f() {
    myClass m;
}

int fun() {
    i = 3;
    f();

    return i++;
}

int main() {
    cout << fun() << " ";
    cout << i << endl;

    return 0;
}
```

What will be the output?

a) `1 5`

b) `3 4`

c) `5 6`

d) `3 5`

Answer: c)
Explanation:
`i` is initialized to 0 (`i = 0;` executes before `main()` is called).
Then `main()` starts and calls `fun()` which sets `i` to 3 by `i = 3;`. The `fun()` calls the function `f()`. In function `f()`, `myClass m;` set `i = 1`. But the object `m` is local within function `f()`. So, as the function `f()` returns, the destructor of local object `m` will be called before return and `i` become 5. This 5 will be returned by `fun()`, and get printed.
Further `i` value incremented to 6 after return (Since `i++` is post incrementer). Finally, value of `i` is printed as 6.
Hence correct option is c).

Question 3

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

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

class Data {
    int x;
    void fun1() {
        cout << "inside fun1";
    }
public:
    int y;
    void fun2() {
        cout << "inside fun2";
    }
};

int main() {
    Data t;
    t.x = 5;  // LINE-1
    t.fun1(); // LINE-2
    t.y = 8;  // LINE-3
    t.fun2(); // LINE-4

    return 0;
}
```

Which line/lines will give error?

a) `LINE-1`

b) `LINE-2`

c) `LINE-3`

d) `LINE-4`

Answer: a), b)
Explanation:
All the class members declared under public accessible inside and outside of the class. The class members declared as private can be accessed only by the `public` functions inside the class.
`LINE-1` gives error as `x` is `private` member.
`LINE-2` gives error as `fun1()` is `private` member.
`LINE-3` is fine as `y` is `public` member.
`LINE-4` is fine as function `fun2()` is public member.

Question 4

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

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

class MyClass {
public:
    MyClass() { cout << "1"; }
    MyClass(const MyClass &t) { cout << "2"; }
};

int main() {
    MyClass *t1, *t2;        // LINE-1

    t1 = new MyClass();      // LINE-2
    t2 = new MyClass(*t1);   // LINE-3

    MyClass t3 = *t1;        // LINE-4
    MyClass t4 = t3;         // LINE-5

    return 0;
}
```

What will be the output?

a) `111222`

b) `1112`

c) `1212`

d) `1222`

Answer: d)
Explanation:
At LINE-1, the statement `MyClass *t1, *t2;` declares two pointers. So no objects are instantiated, hence the constructors are not called.
At LINE-2, the statement `t1 = new MyClass();` invokes the default constructor. Hence the output is 1.
At LINE-3, the statement `t2 = new MyClass(*t1);` invokes the copy constructor. Hence the output is 2.
At LINE-4 and 5, the statements `MyClass t3 = *t1;` and `MyClass t4 = t3;` invoke the copy constructor. Hence the output will be 22.
So option d) is correct.

Question 5

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

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

class MyClass {
    char _____________; // LINE-1: declare the data members
public:
    MyClass(char* _fname, char* _mname, char* _lname) :
        fname(setFname(_fname)), mname(setMname(_mname)),
        lname(setLname(_lname)) { }
    char* setFname(char* fn) {
        cout << fn << " ";
        return strdup(fn);
    }
    char* setMname(char* mn) {
        cout << mn << " ";
        return strdup(mn);
    }
    char* setLname(char* ln) {
        cout << ln << " ";
        return strdup(ln);
    }
};

int main() {
    MyClass obj("Ram", "Mohan", "Roy");

    return 0;
}
```

Fill in the blank at LINE-1 such that the output is as follows:
`Roy Mohan Ram`

a) `*lname, *fname, *mname`

b) `*mname, *lname, *fname`

c) `*fname, *lname, *mname`

d) `*lname, *mname, *fname`

Answer: d)
Explanation:
The order of invocation to initialization-list function depends on the sequence of the data members declared in the class.

Question 6

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

```
class Test {
    // code...
};

int main() {
    const Test t; // LINE-1
    return 0;
}
```

What is the type of **this** pointer associated with the object `t`?

a) `const Test* this;`

b) `Test* const this;`

c) `Test const* const this;`

d) `const Test* const this;`

Answer: c), d)
Explanation:
`this` pointer is always a constant. So for class `Test`, the type of `this` for `Test t` would be `Test * const`.
In **LINE-1**, the object is a constant. So the type of the `this` pointer of a constant object (as specified `const Test`) of `class Test` is:
`const Test* const this;` or `Test const* const this;`

Question 7

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

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

class Test {
    int _x;
    int _y;
    Test(int x, int y) {
        _x = x;
        _y = y;
        cout << _x << " " << _y;
    }
};

int main() {
    Test t(5, 6);

    return 0;
}
```

What will be the output / error?

a) `0 0`

b) `5 6`

c) `compilation error:  no default constructor`

d) `compilation error:  constructor is private`

Answer: d)
Explanation:
The parametrized (and only) constructor `Test(int x, int y)` is private by default. So it cannot be accessed from `main()` function. Hence option d) is correct.

Question 8

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

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

class Data {
    int _d;
public:
    int set_d(int d) const {
        _d = d;
    }
    int get_d() const {
        return _d;
    }
};

int main() {
    Data obj;

    obj.set_d(5);
    cout << obj.get_d();

    return 0;
}
```

What will be the output / error?

a) `0`

b) `5`

c) `compiler error:  assignment of data-member Data::_d is read-only object`

d) `compiler error:  cannot have const function for non-const object`

Answer: c)
Explanation:
As the `set_d()` is a constant function, it cannot change the state of an object (`_d = d;`). Hence when we try to assign value to `_d` it gives compiler error, that is, option c).

Question 9

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

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

class Point {
    int x, y;
public:
    Point(int _x, int _y) : x(_x), y(_y) { }
    void changePoint(Point *new_pt) { this = new_pt; }
    void show() { cout << x << ", " << y << endl; }
};

int main() {
    Point p1(10, 20);
    Point p2(20, 50);

    p1.changePoint(&p2);
    p1.show();

    return 0;
}
```

What will be the output / error?

a) `10, 20`

b) `20, 50`

c) `Compiler Error: lvalue required as left operand of assignment`

d) `Compiler Error: private x, y are inaccessible`

Answer: c)
Explanation:
In the function `changePoint(&p2)`, `this = new_pt;` is an assignment to `this`. Since `this` is a constant pointer `(Point * const)`, it cannot be changed and the error occurs during compilation which asks for an l-value (or address where the content can be changed).

Programming Questions

Question 1

Consider the following program and fill the blanks (in LINE-1, LINE-2, and LINE-3) with appropriate definitions for constructor, destructor and area(). Please check the sample input and output. *Marks: 3*

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

class triangle {
    const int *_base, *_height;
public:
    triangle(int b, int h) : ____________________ { }
    // LINE-1: Complete Constructor definition
    ~triangle() {
        ____________________;
        // LINE-2: Complete destructor to delete both data pointers
    }
    double area();
};

____________________ {  // LINE-3: Complete function header
    return 0.5 * *_base * *_height;
}

int main() {
    int a, b;

    cin >> a >> b;
    triangle r(a, b);
    cout << r.area();

    return 0;
}
```

Public 1

```
Input: 40 50
Output: 1000
```

Public 2

```
Input: 86 5
Output: 215
```

Private

```
Input: 10 20
Output: 100
```

Answer:
LINE-1: `_base(new int(b)), _height(new int(h))`
LINE-2: `delete _base, _height`
LINE-3: `double triangle::area()`

Explanation:
As in the destructor memory allocated for `*base` and `*height` are freed using delete operator, in constructor the allocation of memory must be done dynamically using new operator. Hence we use the constructor with initialization list (at LINE-1) as:
`_base(new int(b)), _height(new int(h))`
LINE-2 can be filled as follows:
`delete _base, _height`
Similarly, in LINE-3 we have defined the area() function outside of the class hence we need to use scope resolution operator as follows:
`double triangle::area()`

Question 2

Consider the following program. Fill in the blanks at `LINE-2` and `LINE-3` to make the function constant and at `LINE-1` to make the variable editable from the constant function. Consider the following test cases. *Marks: 3*

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

class Student {
    int _roll;
    string _name;
    __________ int _sem;        // LINE-1
public:
    Student(int roll, string name, int sem)
        : _roll(roll), _name(name), _sem(sem) { }

    void promote() __________ { // LINE-2
        _sem++;
    }

    void display() __________ { // LINE-3
        cout << "[" << _roll << "] " << _name << " : " << _sem << endl;
    }
};

int main() {
    string a;
    int b, c;

    cin >> a >> b >> c;
    const Student s(b, a, c);

    s.promote();
    s.display();

    return 0;
}
```

Public 1

```
Input: soumen 10 1
Output: [10] soumen : 2
```

Public 2

```
Input: arup 22 3
Output: [22] arup : 4
```

Private

```
Input: himadri 30 6
Output: [30] himadri : 7
```

Answer:

```
LINE-1: mutable
LINE-2: const
LINE-3: const
```

Explanation:
A `mutable` data member `_sem` of class can be accessed as well as updated even within a `const` member function for `const` objects of the same class.

Question 3

Consider the following program and fill in the blanks at `LINE-1, LINE-2, LINE-3` with appropriate initialization of data-members such that it satisfies the given test cases.

Marks: 3

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

class Complex {
    const int _r, _i;
public:
    Complex() : __________ { }                      // LINE-1
    Complex(int r) : __________ { }                 // LINE-2
    Complex(int r, int i) : __________ { }          // LINE-3
    void show() { cout << _r << " + " << _i << "i" << endl; }
    int norm() { return _r * _r + _i * _i; }
};

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

    Complex c1;
    Complex c2(a);
    Complex c3(a, b);

    c1.show();
    c2.show();
    c3.show();

    cout << c1.norm() << ", " << c2.norm() << ", " << c3.norm();

    return 0;
}
```

Public 1

```
Input: 3 5
Output:
0 + 0i
3 + 0i
3 + 5i
0, 9, 34
```

Public 2

```
Input: 30 10
Output:
0 + 0i
30 + 0i
30 + 10i
0, 900, 1000
```

Private

```
Input: 4 10
Output:
0 + 0i
4 + 0i
4 + 10i
0, 16, 116
```

Answer:

```
LINE-1:  _r(0), _i(0)
LINE-2:  _r(r), _i(0)
LINE-3:  _r(r), _i(i)
```

Explanation:
At LINE-1, default constructor need to initialize `const` data members with 0. Hence, the code will be:
`Complex() :  _r(0), _i(0) { }`
At LINE-2, the parametrized constructor initializes the data member `_r` with the given value, and `_i` with 0. Hence, the code will be:
`Complex(int r) :  _r(r), _i(0) { }`
At LINE-3, the parametrized constructor initializes the data members `_r, _i` with the given value. Hence, the code will be:
`Complex(int r, int i) :_r(r), _i(i) { }`

Question 4

Fill in the blanks as instructed to declare and define appropriate objects so that the program generates the output as per the given test cases. *[Marks 3]*

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

class integer {
    int _x;
public:
    integer() : _x(0) { }
    integer(int x) : _x(x) { }
    integer(const integer &obj) : _x(obj._x) { }
    void display() { cout << _x << " "; }
};

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

    _____________;  // LINE-1: Invoke Default Constructor
    _____________;  // LINE-2: Invoke Parameterized Constructor
    _____________;  // LINE-3: Invoke Copy Constructor

    objA.display();
    objB.display();
    objC.display();

    return 0;
}
```

Public 1

```
Input: 25
Output: 0 25 25
```

Public 2

```
Input: 16
Output: 0 16 16
```

Private

```
Input: 30
Output: 0 30 30
```

Answer:

```
LINE-1:  integer objA

LINE-2:  integer objB(val)
```

Or

```
LINE-2:  integer objB = val

LINE-3:  integer objC(objB)
```

Or

```
LINE-3:  integer objC = objB
```

Explanation:
Given

```
integer(): _x(0) { }                          // Default Constructor
integer(int x): _x(x) { }                     // Parameterized Constructor
integer(const integer &obj): _x(obj._x) { } // Copy Constructor
```

Note the different syntax and order for invoking different constructors.