Programming in Modern C++: Assignment Week 3

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 15, 2025

Question 1

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

```
class Complex{
    private:
        int re, im;
    public:
        void setRE(int r_){ re = r_; }
        void setIM(int i_){ im = i_; }
        void print(){ cout << re << ", i" << im; }
        void incr(){ re++, im++; }
        int incrRE(){ return re + 1; }
        int incrIM(){ return ++im; }
};
```

Identify set of all methods that change the state of `Complex` class objects?

a) `setRE(), setIM(), print()`

b) `setRE(), setIM(), incrRE(), incrIM()`

c) `incr(), incrRE(), incrIM()`

d) `setRE(), setIM(), incr(), incrIM()`

Answer: d)
Explanation:
The function `setRE()` changes the data member value `re`. Thus, `setRE()` changes the state of the object.
The function `setY()` changes the data member value `im`. Thus, `setIM()` changes the state of the object.
The function `incr()` changes the values of data members `re` and `im`. Thus, `incr()` changes the state of the object.
The function `incrIM()` changes the value of data member `im`. Thus, `incrIM()` changes the state of the object.
Note that, the function `incrRE()` does not change the value of data member `re`, as it returns an expression only i.e. `(re+1)`.

Question 2

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

```
#include <iostream>
using namespace std;
    class Number {
        int n;
    public:
        Number(){
            cout << 0 << " ";
        }
        Number(int i): n(i) {
            cout << n << " ";
        }
};
int main() {
    int i = 1;
    Number n1();    //LINE-1
    Number *n2 = new Number(i++);
    Number *n3;
    new Number(i++);
    return 0;
}
```

What will be the output?

a) `0 1 0 2`

b) `0 1 2`

c) `0 2 3`

d) `1 2`

Answer: d)
Explanation:
The statement `Number n1();` is not an error, but it does not instantiate an object.
Statement `Number *n2 = new Number(i++);`, instantiate an object, and call parameterized constructor with the value of `i` i.e. 1. Hence it prints 1.
Statement `Number *n3;`, just create a pointer, don't instantiate an object.
Statement `new Number(i++);`, creates a temporary object, and call the parameterized constructor with the value of current `i` i.e. 2. Hence prints 2.
Hence, the correct option is d).

Question 3

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

```
#include <iostream>
#include <cstring>
using namespace std;
    class MyClass {
        const char _______________; // LINE-1: declare the data members
    public:
        MyClass(const char* _s1, const char* _s2, const char* _s3) :
        s1(setS1(_s1)), s2(setS2(_s2)),
        s3(setS3(_s3)){}
        const char* setS1(const char* s) {
            cout << s << " ";
            return strdup(s);
        }
        const char* setS2(const char* s) {
            cout << s << " ";
            return strdup(s);
        }
        const char* setS3(const char* s) {
            cout << s << " ";
            return strdup(s);
        }
};
int main() {
    MyClass obj("programming", "in", "C++");
    return 0;
}
```

Fill in the blank at `LINE-1` such that the program will print `in C++ programming` ?

a) `*s2, *s3, *s1`

b) `*s1, *s2, *s3`

c) `*s1, *s3, *s2`

d) `*s2, *s1, *s3`

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

Question 4

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

```
#include <iostream>
#include <string>
using namespace std;
class Test{
    int _t;
    public:
        int set_t(int t) const {
            _t = t;
        }
        int get_t() const {
            return _t;
        }
};
int main(){
    Test obj;
    obj.set_t(5);
    cout<<obj.get_t();
    return 0;
}
```

What will be the output/error?

a) 0

b) 5

c) Compiler error: assignment of data-member Test::_t is read-only object

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

Answer: c)
Explanation:
As the set_t() is a constant function, it cannot change the state of an object. Hence when we try to assign a value to _t (a data member), it gives compiler error, i.e. option c).

Question 5

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

```
#include <iostream>
using namespace std;
class Complex {
    int re, im;
    public:
        Complex(int _re, int _im) : re(_re), im(_im) { }
        void change(Complex *new_C) { this = new_C; }
        void show() { cout << re << " + i" << im << endl; }
};
int main() {
    Complex c1(10, 20);
    Complex c2(20, 50);
    c1.change(&c2);
    c1.show();
    return 0;
}
```

What will be the output/error?

a) `10 + i20`

b) `20 + i50`

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

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

Answer: c)
Explanation:
In the function `c1.change(&c2)`, the statement
`this = new_C;`
attempts to make assignment to `this`. Since `this` is a constant pointer `(Complex * const)`, it cannot be changed and the error occurs during compilation.

Question 6

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

```
class myClass {
    // code...
};
int main() {
    const myClass m; // LINE-1
    return 0;
}
```

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

a) `const myClass* this;`

b) `myClass* const this;`

c) `myClass const* const this;`

d) `const myClass* const this;`

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

Question 7

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

```
#include<iostream>
using namespace std;
class Data {
    public:
        Data() { cout << "O"; }                     //LINE-1
        Data(Data *t) { cout << "K"; }              //LINE-2
        Data(const Data &t) { cout << "Z"; }        //LINE-3
};
int main(){
    Data *t1, *t2;
    t1 = new Data();
    t2 = new Data(t1);
    Data t3 = *t1;
    Data t4 = t3;
    return 0;
}
```

What will be the output?

a) `OKKK`

b) `OKZZ`

c) `OKKZ`

d) `OZZZ`

Answer: b)
Explanation:
The constructor defined in `LINE-1` is a default constructor which is invoked when the statement `t1 = new Data();` is executed.
A parameterized constructor is defined in `LINE-2` which is invoked when `t2 = new Data(t1)` is executed.
The copy constructor in `LINE-3` is invoked twice for `t3` and `t4` initialization.

Question 8

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

```
#include<iostream>
using namespace std;
class String {
    char x;
    public:
        String(char _x): x(_x) { }
        void display() { cout << __________ << " "; } //LINE-1
};
int main() {
    String c('C');
    c.display();
    return 0;
}
```

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

a) `++this->x`

b) `++this.x`

c) `++x`

d) `x++`

Answer: a), c)
Explanation:
When the `display(.)` function is called, the value of `x` is "C". So, we need to increment `x` before printing. It can be done using `++this->x or ++x`.

Question 9

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

```
#include<iostream>
using namespace std;
static int i = 5;
class myClass {
    public:
        myClass() { cout << ++i; }
        ~myClass() { cout << i--; }
};
void check(myClass c){
    //Some Code
}
int main() {
    myClass c1;
    check(c1);
    return 0;
}
```

What will be the output?

a) `5665`

b) `555`

c) `665`

d) `6565`

Answer: c)
Explanation:
The lifetime of static variable is present throughout the program. When the object `c1` is declared, `i` is incremented by 1 and printed from the default constructor. After that, the function `check(.)` is called with `c1` as call-by-value parameter which copies the whole object to the actual parameter c of function `check(.)`. So, default copy constructor is called. After the function lifetime, the actual parameter is destroyed which calls the class destructor. So, the value of i is printed i.e. 6 then decrement it. After the execution ends, destructor of main function object `c1` calls which prints the value of i again i.e. 5. So, correct option is c.

Programming Questions

Question 1

Consider the program below which defines a class `Complex`.
Complete the program with the following instructions.

- Fill in the blank at `LINE-1` to complete parameterized constructor.
- Fill in the blank at `LINE-2` to complete copy constructor.
- Fill in the blank at `LINE-3 and LINE-4` to complete the `sum` function.

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

```
#include<iostream>
#include<cmath>
using namespace std;
class Complex{
    const int x,y;
public:
    Complex(int _x=0, int _y=0) : ____________ {} //LINE-1
    Complex(const Complex& c) : ______________ {} //LINE-2
    void sum(Complex p){
        int rx = _____; //LINE-3
        int ry = _____; //LINE-4
        cout << "(" << rx << "," << ry << ")" << endl;
    }
    void print(){ cout << "(" << x << "," << y << ")" << endl; }
};
int main(){
    int x1,x2,y1,y2;
    cin >> x1 >> y1 >> x2 >> y2;
    Complex c1(x1,y1), c2(x2,y2);
    c1.print();
    c2.print();
    c1.sum(c2);
    return 0;
}
```

Public 1

```
Input: 1 2 3 4
Output:
(1,2)
(3,4)
(4,6)
```

Public 2

```
Input: 5 10 15 20
Output:
(5,10)
(15,20)
(20,30)
```

Private 1

```
Input: 2 4 5 8
Output:
(2,4)
(5,8)
(7,12)
```

Answer:

```
LINE-1:  x(_x), y(_y)
LINE-2:  x(c.x), y(c.y)
LINE-3:  x+p.x OR p.x+x
LINE-4:  y+p.y OR p.y+y
```

Explanation:

The parameterized constructor can be completed at `LINE-1` with the initializer as `x(_x), y(_y)`. Similarly, the copy constructor can be completed at `LINE-2` as `x(p.x), y(p.y)`. The `sum` function at `LINE-3` and `LINE-4` can be computed as `x+p.x` and `y+p.y`.

Question 2

Consider the following program.

- Fill in the blanks at `LINE-1` and `LINE-2` with an appropriate constructor and destructor statement.
- Fill in the blank at `LINE-3` with appropriate header for assignment overload function.
- Fill in the blank at `LINE-4` with an appropriate concatenation statement.

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

```
#include<iostream>
#include<malloc.h>
#include<string.h>
using namespace std;
class Test{
    char *s;
public:
    Test(char *s) : ____________ {} //LINE-1
    ~Test(){ _________ }           //LINE-2
    ______________________________{ //LINE-3
        free(s);
        s = strdup(m.s);
        return *this;
    }
    void update(char* x){
        ______________________;     //LINE-4
    }
    void print(){
        cout << s << endl;
    }
};
int main(){
    string str1, str2;
    cin >> str1 >> str2;
    Test *m1 = new Test(&str1[0]);
    Test *m2 = m1;
    m2->update(&str2[0]);
    m2->print();
    delete(m1);
    return 0;
}
```

Public 1

```
Input: Hello Sir
Output: Hello Sir
```

Public 2

```
Input: Good Night
Output: Good Night
```

Private

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

Answer:

```
LINE-1:  s(strdup(s))
LINE-2:  free(s);
LINE-3:  Test& operator=(const Test& m)
LINE-4:  strcat(strcat(s," "),x)
```

Explanation:

The constructor at `LINE-1` can be filled as `s(strdup(s))`. Similarly, destructor is used to free the dynamically allocated memory. So, `LINE-2` will be filled as `free(s)`. The operator header at `LINE-3` will be filled as `Test& operator=(const Test& m)`. `LINE-4` is used to concatenate the parameter string with the class data member along with a space in the middle. So, `LINE-4` will be filled as `strcat(strcat(s," "), x)`.

Question 3

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

- at LINE-1 with appropriate declaration of data member z,
- at LINE-2 with appropriate constructor statement, and
- at LINE-3 and LINE-4 with appropriate header of the functions calcZ() and print(),

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

```
#include<iostream>
using namespace std;
class Point3D {
    int x, y;
    __________________; // LINE-1
public:
    Point3D(int x_, int y_) : ______________________ { } //LINE-2
    _____________________ { z = x * y; }; // LINE-3
    _____________________ { // LINE-4
        cout << "(" << x << "," << y << "," << z << ")";
    }
};
int main() {
    int i, j;
    cin >> i >> j;
    const Point3D m(i, j);
    m.calcZ();
    m.print();
    return 0;
}
```

Public 1

```
Input: 3 5
Output: (9,25,225)
```

Public 2

```
Input: 10 -5
Output: (100,25,2500)
```

Private

```
Input: 20 10
Output: (400,100,40000)
```

Answer:

```
LINE-1:  mutable int z
LINE-2:  x(x_ * x_), y(y_ * y_)
LINE-3:  void calcZ() const
LINE-4:  void print() const
```

Explanation:

Since `m` is defined as a constant object, and we need to modify the value of `z`, `z` has to be defined as `mutable` member. Thus, the declaration of `z` can be as follows:

```
LINE-1:  mutable int z
```

Since the functions `calcZ()` and `print()` are called on a constant object, they must be defined as constant functions as follows:

```
LINE-2:  void calcZ() const
LINE-3:  void print() const
```

And according to the test cases, `LINE-2` will be filled as

```
LINE-2:  x(x_ * x_), y(y_ * y_)
```