Programming in Modern C++: Assignment Week 7

Total Marks : 25

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

March 2, 2023

Question 1

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

```
#include <iostream>
using namespace std;
class employee {
    string name ;
    int salary;
    public:
        employee(int _sal, string _name) : name(_name), salary(_sal) {}
        void update(int s, string na) const{
            ( ________________________ )->salary = s; //LINE-1
            ( ________________________ )->name = na;  //LINE-2
        }
        void showInfo() const {
            cout << name << " : " << salary;
        }
};
int main(void) {
    const employee e(3000, "Raj");
    e.update(5000, "Rajan");
    e.showInfo();
    return 0;
}
```

Fill in the blank at `LINE-1` and `LINE-2` with the same statement such that the program will print `Rajan :  5000`.

a) `const_cast <employee*> (this)`

b) `static_cast <employee*> (this)`

c) `dynamic_cast <employee*> (this)`

d) `(employee*)(this)`

Answer: a), d)
Explanation:

The statement `const employee e(3000, "Raj");` defines `e` as a constant object. To modify its data-members, the constant-ness of the object need to be removed. This can be done either by `const_cast` (in option a) or casting constant this pointer to `employee*` type (in option d).

Question 2

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

```
#include<iostream>
using namespace std;
class A{
    public:
        virtual void f() {}
        virtual void g() {}
};
class B : public A{
    public:
        void g() {}
        void h() {}
        virtual void i();
};
class C : public B{
    public:
        void f() {}
        virtual void h() {}
};
```

What will be virtual function table (VFT) for the `class C`?

a) `C::f(C* const)`
   `B::g(B* const)`
   `C::h(C* const)`
   `B::i(B* const)`

b) `A::f(A* const)`
   `B::g(B* const)`
   `C::h(C* const)`
   `B::i(B* const)`

c) `A::f(A* const)`
   `B::g(B* const)`
   `B::h(B* const)`
   `C::i(C* const)`

d) `A::f(A* const)`
   `B::g(B* const)`
   `C::h(C* const)`
   `C::i(C* const)`

Answer: a)
Explanation:
All four functions are virtual in the `class C`. So, there will be four entries in virtual function table.
Now, function `f()` is overridden in `class C`. So, the entry for function `f()` in the virtual function table of `class C` will be `C::f(C* const)`.
The function `g()` is virtual from class A and is overridden in `class B`. So, the entry for function `g()` in VFT of `class C` will be `B::g(B* const)`.
The function `h()` is declared as virtual in `class C`. So, the entry for function `h()` in VFT of class C will be `C::h(C* const)`.

The function `i()` is declared as virtual in `class B`. But not overridden in `C`. So, the entry for function `i()` in VFT of `class C` will be `B::i(B* const)`.

Question 3

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

```
#include <iostream>
using namespace std;
int main() {
    char c = 'C';
    double d = 3.14;
    char *cp = &c;
    double *pd;
    c = static_cast<char>(d);       // LINE-1
    d = static_cast<double>(c);     // LINE-2
    pd = static_cast<double*>(cp); // LINE-3
    c = static_cast<char>(&c);      // LINE-4
    return 0;
}
```

Which line/s will give compilation error?

a) `LINE-1`

b) `LINE-2`

c) `LINE-3`

d) `LINE-4`

Answer: c), d)
Explanation:
`static_cast` cannot cast between two different pointer types. In `LINE-3`, `double*` is assigned to `char*`. Hence it is error.
Using static cast, it is not possible to change a pointer type to a value type. In `LINE-4`, `char*` is assigned to `char` which is not possible using static cast.

Question 4

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

```
class Test1 { };
class Test2 { };
Test1* t1 = new Test1;
Test2* t2 = new Test2;
```

Which of the following type-casting is permissible?

a) `t2 = static_cast<Test2*>(t1);`

b) `t2 = dynamic_cast<Test2*>(t1);`

c) `t2 = reinterpret_cast<Test2*>(t1);`

d) `t2 = const_cast<Test2*>(t1);`

Answer: c)
Explanation:
On each option, there is an attempt to cast from `Test1*` to `Test2*`, and these two classes are unrelated. As we know, only `reinterpret_cast` can be used to convert a pointer to an object of one type to a pointer to another object of an unrelated type. Hence only option c) is correct.

Question 5

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

```
#include <iostream>
#include <typeinfo>
using namespace std;
class B { public: virtual ~B(){}};
class D: public B {};
int main() {
    B b;
    D d;
    D *dp = &d;
    B *bp = dp;
    D *dpp = (D*)dp;
    cout << (typeid(bp).name() == typeid(dpp).name());
    cout << (typeid(*bp).name() == typeid(*dpp).name());
    cout << (typeid(dp).name() == typeid(dpp).name());
    cout << (typeid(*dp).name() == typeid(*dpp).name());
    return 0;
}
```

What will be the output?

a) `0101`

b) `0111`

c) `0110`

d) `0010`

Answer: b)
Explanation:
Type of `bp` is `B*` and type of `dpp` is `D*`. Thus, output is 0.
`*bp` and `*dpp` point to the same object d, and it is a dynamic binding situation. Thus, both are of type D and output is 1.
Type of `dp and dpp` is `D*`. Thus, output is 1.
`*dp and *dpp` point to the same object d, and it is a dynamic binding situation. Thus, both are of type D and output is 1.

Question 6

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

```
#include <iostream>
using namespace std;
class A{ public: virtual ~A(){} };
class B : public A{};
class C : public A{};
int main(){
    A objA;
    B objB;
    A* pA = dynamic_cast<A*>(&objB); //LINE-1
    pA == NULL ? cout << "0" : cout << "1";
    B* pB = dynamic_cast<B*>(pA); //LINE-2
    pB == NULL ? cout << "0" : cout << "1";
    C* pC = dynamic_cast<C*>(new A); //LINE-3
    pC == NULL ? cout << "0" : cout << "1";
    pC = dynamic_cast<C*>(&objB); //LINE-4
    pC == NULL ? cout << "0" : cout << "1";
    return 0;
}
```

What will be the output?

a) `0101`

b) `1010`

c) `1100`

d) `1011`

Answer: c)
Explanation:
The type-casting at `LINE-1` is valid as it is an upper-casting.
At `LINE-2`, though it is a down-casting, it is allowed as the pointer `pB` points to the same type of object (which of type B).
At `LINE-3`, the down-casting is invalid as the pointer pC points to parent type of object (which is of type A). Hence prints 0.
At `LINE-4`, the casting is also invalid as the pointer pC points to an object (which is of type B) that is neither of its base type or derived type and hence prints 0.

Question 7

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

```
#include <iostream>
using namespace std;
int main() {
    const double g = 9.8;
    const double *pg = &g;
    double *pt = ________________________(pg); //LINE-1
    *pt = 9.81;
    cout << *pt;
    return 0;
}
```

Fill in the blank at `LINE-1` so that the program will print `9.81`.

a) `const_cast<double*>`

b) `static_cast<double*>`

c) `dynamic_cast<double*>`

d) `(const double*)`

Answer: a)
Explanation:
At `LINE-1`, pg of type `const double*` needs to be casted to `double*`. It can be accomplished by `const_cast<double*>(pg)`.
This question is intentionally made as MSQ

Question 8

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

```
#include<iostream>
using namespace std;
class C1{
    string a = "C++";
};
class C2{
    public:
        string b = "Programming";
};
int main(){
    C1 u;
    C2 *v = ____________________________(&u);
    cout << v->b;
    return 0;
}
```

Fill in the blank at `LINE-1` so that the program will print "C++".

a) `reinterpret_cast<C2*>`

b) `static_cast<C2*>`

c) `dynamic_cast<C2*>`

d) `(C2*)`

Answer: a), d)
Explanation:
We need to cast object of `C1` to `C2` type which are unrelated. This can be done using C style casting or reinterpret cast.

Question 9

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

```
class A1{ public: void f(){} };
class A2 : public A1 { public: virtual void f(){}};
class A3 : public A2{ public: void g(){} };
class A4 : public A1{ public: virtual void g(){} };
```

How many virtual function table (VFT) will be created?

a) `1`

b) `2`

c) `3`

d) `4`

Answer: c)
Explanation:
The presence of a virtual function (either explicitly declared or inherited from a base class) makes the class polymorphic. For such classes, we need a class-specific virtual function table (VFT). All three classes except `A1`, thus, will set up virtual function tables.

Programming Questions

Question 1

Complete the program with the following instructions.

- Fill in the blank at `LINE-1` to complete operator overloading for assignment operator.
- Fill in the blanks at `LINE-2` and `LINE-3` to complete the type casting statements.

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

```
#include<iostream>
using namespace std;
class Test1{
    int a = 10;
    public:
        void show(){
            cout << a << " ";
        }
};
class Test2{
    int b = 20;
    public:
        void show(){
            cout << b;
        }
        ______________________{ //LINE-1
            b = b + x;
        }
};
void fun(const Test1 &t, int x){
    Test1 &u = ____________________(t); //LINE-2
    u.show();
    Test2 &v = ___________________________(u); //LINE-3
    v = x;
    v.show();
}
int main(){
    Test1 t1;
    int x;
    cin >> x;
    fun(t1, x);
    return 0;
}
```

Public 1

```
Input: 4
Output: 10 14
```

Public 2

```
Input: 9
Output: 10 19
```

Private 1

```
Input: 15
Output: 10 25
```

Answer:

```
LINE-1:  operator=(int x)
LINE-2:  const_cast<Test1&>
LINE-3:  reinterpret_cast<Test2&>
```

Explanation:

As per the function `fun()`, we need to overload the operator equal to for the class `Test2` at `LINE-1` so that the assignment `v = x` will be valid. It can be done as `operator=(int x)`.

To call a non-constant function `show()` using a const object reference u, we need to cast the reference to a non-const reference. So, `LINE-2` will be filled as `const_cast<Test1&>`.

Casting between two unrelated classes at `LINE-3` can be done as `reinterpret_cast<Test2&>`.

Question 2

Consider the following program with the following instructions.

- Fill in the blank at `LINE-1` to complete constructor definition.
- Fill in the blank at `LINE-2` to complete assignment operator overload function signature.
- Fill in the blank at `LINE-3` to complete integer cast operator overload function signature.

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

```
#include<iostream>
using namespace std;
class intString{
    int *arr;
    int n;
    public:
        intString(int k) : ________________________{} //LINE-1
        ________________{ //LINE-2
            return arr[--n];
        }
        ___________________(int &k){ //LINE-3
            int t;
            for(int j = 0; j < k; j++){
                cin >> t;
                this->arr[j] = t;
            }
            return *this;
        }
};
int main(){
    int k;
    cin >> k;
    intString s(k);
    s = k;
    for(int i = 0; i < k; i++)
        cout << static_cast<int>(s) << " ";
    return 0;
}
```

Public 1

```
Input: 3
1 2 3
Output: 3 2 1
```

Public 2

```
Input: 4
5 6 4 7
Output: 7 4 6 5
```

Private

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

Answer:

```
LINE-1:  n(k), arr(new int(n))
LINE-2:  operator int()
LINE-3:  intString operator=
```

Explanation:

The initialization of the data-members at `LINE-1` can be done as:

```
n(k), arr(new int(k))
```

At `LINE-2`, we overload type-casting operator for the statement `static_cast<int>(s)` as:

```
operator int()
```

At `LINE-3`, we overload `operator=` for the statement `s = k;` as:

```
intString operator=(int& k)
```

Question 3

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

- at `LINE-1` to complete the constructor definition,
- at `LINE-2` to complete the function header to overload operator for type-casting to `char`,
- at `LINE-3` to complete the function header to overload operator for type-casting to `int`,

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

```
#include<iostream>
#include<cctype>
using namespace std;
class classChar{
    char ch;
    public:
        _____________ : ch(tolower(_ch)){} //LINE-1
        _____________{ return ch; } //LINE-2
        _____________{ return ch - 'a' + 1; } //LINE-3
};
int main(){
    char c;
    cin >> c;
    classChar cb = c;
    cout << (char)cb << ": position is " << int(cb);
    return 0;
}
```

Public 1

```
Input: A
Output: a: position is 1
```

Public 2

```
Input: c
Output: c: position is 3
```

Private

```
Input: Z
Output: z: position is 26
```

Answer:
`LINE-1: classChar(char _ch)`
`LINE-2: operator char()`
`LINE-3: operator int()`
Explanation:
The statement `Char cb = c;` requires casting from char to class `classChar`, which is implemented by the constructor as:
`classChar(char _ch)`
The statement `(char)cb` requires a casting operator from class `classChar` to char which may be done by the function

`operator char()`
The statement `int(cb)` requires a casting operator from class `classChar` to int which may be done by function
`operator int()`