Khoi Duong

Prof. Yang

CS360L

8/16/2022

FINAL

1.

a. Inheritance
b. Public and protected
c. Inheritance
d. Composition or aggregation
e. Hierarchical
f. Public
g. Private
h. Multiple inheritance
i. constructor
j. Public, protected
k. Protected, protected

2.

a. True
b. False. A has-a relationship is inherited via composition. A is-a relationship is inherited via inheritance

c. False. A Car class has *has-a* relationship with the SteeringWheel and Brakes classes

d. True

e. True

3.

a. Derived-class object

b. Switch

c. Abstract

d. Concrete

e. dynamic_cast

f. type_info

g. Polymorphism

h. Virtual

i. Downcasting

4.

a. False. An abstract base class can include virtual methods with implementations.

b. False. Referring to a derived-class object with a base-class handle is normal. Referring to a base-class object with derived-class handle is dangerous.

c. False. A class is abstract by making at least one pure virtual method in the class. We don't declare class as virtual.

d. True

e. True

5.

a. False. Keywords typename and class also allow for a type parameter of a fundamental type.
b. True
c. False. Template parameter names among template definitions don't need to be unique.
d. True

6.

a. Function-template specializations, class-template specializations
b. template, angle brackets (< >)
c. overload
d. generic
e. Scope resolution

7.

a. Missing brackets for x <= y. The compiler will understand it as an overloading operator<=, thus creating an error.

   Fix: *cout << "Value of x <= y is: " << (x <= y);*
b. This will cout the character 'c'

   Fix: *cout << int('c');*
c. This will create an error as the quotes using for the display and the quotes using for string input are the same.

   Fix: cout << " 'A string in quotes' ";

8.

a. The code will print 12345. It sets the output width to 5, and fill the space with "*" for the next output stream, which is 123. Thus, the code will print "**123" next. After 'endl;', the output stream resets and it prints '123'.

   Final output:

```
12345
**123
123
```

b. The code will set the output stream's width to 10, and fill the space left by the next output with '$'. The next output is 10000, therefore, it will print '$$$$$10000'

   Final output:
```
$$$$$10000
```

c. The code will set the output stream's width to 8 and the output will set the decimal precision to hundredth (1/100), which means 2 digits after the point.

   Final output:
```
1.02e+03
```

d. The code will output number 99 under oct base format, and then under hex base format.

   Final output:
```
0143
0x63
```

e. The code will output 100000 and then sets the showpos flag to the next output, which means it will add the positive sign (+) if the next output is positive. It prints +100000

   Final output:
```
100000
+100000
```

f. The code will set the output stream's width to 10, set the decimal precision to hundredth (1/100), and change the output format into scientific format.

   Final output:
```
4.45e+02
```

9.

Source code:

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

template <class Type>
bool isEqualTo(Type a ,Type b)
{
    if(a==b)
    return true;
    return false;
}

int main ()
{int a=1,b=2,f=2;
double c=3,d=4,e=4;
char g='a',h='b';
cout<<"a=1, b=2: "<<isEqualTo(a,b)<<endl;
cout<<"b=2, f=2: "<<isEqualTo(b,f)<<endl;
cout<<"c=3, d=4: "<<isEqualTo(c,d)<<endl;
cout<<"d=4, e=4: "<<isEqualTo(d,e)<<endl;
cout<<"h='b', g='a': "<<isEqualTo(h,g)<<endl;

return 0;
}
```

Run program & result:

```
a=1, b=2: 0
b=2, f=2: 1
c=3, d=4: 0
d=4, e=4: 1
h='b', g='a': 0
```

If we don't overload the equality operator, we don't have any operator to compare the two variables. Thus, the compiler cannot run the program. When we overload the equality operator== the compiler now can process the code and compare the two variables.