VE280 2022FA RC5

L14: Subtypes; Inheritance; Virtual Functions

Subtypes

Definition: S is called the subtype of T whenever the instance of an object of type T is expected, an object of type S can be supplied.

Question: Whether the following type showing in left is the subtype of that is showing right?

```
double, int
ifstream, istream
```

How to create subtypes?

1. Add one or more operations.

   Example: `IntSet` and `MaxIntSet`, where `MaxIntSet` has an addtional function `max()`. So, `MaxIntSet` is the subtype of `InSet`.

2. Strengthen the postcondition of one or more operations.

   Remark: What are called postcondition here are "EFFECTS" and return type.

   Example:

   ```
   int A::f();
       //EFFECTS: return a int.
   int B::f();
       //EFFECTS: return a positive int.
   int main(){
       int integer = A::f();
       //Since the variable need A() to return an int, it is obviously legal to
   return a positive int. So B::f() can replace A::f() here.
       //However, if we actually need a positive integer, B::f() can not be
   replaced by A::(), since an integer is not always a positive integer.
   }
   ```

3. Weaken the precondition of one or more operations.

   Remark: What are called precondition here are "REQUIRES" and argument type.

Example:

```
void A::f(string a);
    //REQUIRES: the input string a should be a non-empty string.
void B::f(string b);
    //No specific limitation to b
int main(){
    string str("lkz");
    A::f("lkz");
    //Since the input of A::f() should be non-empty, it will always work if
we replace A::f() with B::f(), since B::f() does not have such limitations.
    //So B is the subtype of A.
}
```

Creating Subtypes by C++

Inheritance: Create a subclass of the base class

```
//Syntax
class son : public father{};
//use ":" to show the inheritance relationship.
//use "public", "private", "protected" to specify the type of inheritance
```

Base Class Member Type	Inheritance Type	Derived Class Member Type
private	public, private, protected	No permission
public	public, private, protected	public, private, protected respectively
protected	public, protected	protected
protected	private	private

Remark: If you want the member of the base class to be accessed by the derived class, but you do not want the outsider to access the member of the bass class, then use protected.

Note: Subclasses may not be subtypes.

Virtual Functions

Motivation: Consider the following situations:

```
//base class father has three children (subclasses)
class father{
    public:
    void speak(){
        cout<<"I am Eddard Stark"<<endl;
    }
};
class son1: public father{
    void speak(){
```

```
        cout<<"I am John Snow"<<endl;
    }
}
class son2: public father{
    void speak(){
        cout<<"I am Arya Stark"<<endl;
    }
}
class son3: public father{
    void speak(){
        cout<<"I am Robb Stark"<<endl;
    }
}
//Then we want to use the pointer of the father type to represent all the three
subtypes as the function input.
void loudspeak(father * role){
    *role.speak();
    //In this domain, however, the role can only be seen as an instance of
"father", but you really want to use function of speak() of son1, son2 or son3.
}

int main(){
    son1 John;
    son2 Arya;
    son3 Robb;
    loudspeak(&John);//This will work due to the replacement principle of
subtype.
    loudspeak(&Arya);
    loudspeak(&Robb);

}
```

Then, we can define the corresponding function in the father type as ***virtual***.

```
class father{
    public:
    virtual void speak(){
        cout<<"I am Eddard Stark"<<endl;
    }
};
```

This time, when the *role.speak() is called, the compiler will strive to find the real type of role, not just the base type.

L15 Interfaces; Invariants

Motivation: To be consistent with the definition of ADT, we provide an "interface-only" class as a base class (Abstract Base Class). In the ABC, we only provide the declaration of some functions, and we never provide the definition of functions and member types.

```
class car{ //There we only provide an abstraction of the "car".
    virtual void run() = 0; //The car can "run"
    virtual void stop() = 0; //The car can "stop"
}
```

Remark: Function with "= 0" ended is called pure virtual function. The class with pure virtual functions is called abstract class.

Note: You cannot create any instance of an abstract class.

```
class bus{
    int people[10];
    int num;
    int capacity;
    public:
    bus();//constructor
    void run();
    void stop();
}
```

Remark: Typically, you write your interface in a public header file, and write your implementation (the derived class) in a source file. In that case, the user can get how to use the functions through the header file, but cannot get the exactly definition. This perfectly fits our requirements of ADT.

Write the following functions to help user get the instance of bus

```
// header file
car * getCar();

//cpp file
static bus bus210; //This only works for only one instance is needed.
IntSet *getCar(){
    return &bus1;
}
```

For multiple instances are needed, use dynamic memory allocation.

Questions

1. What is the output?

```
#include <iostream>
using namespace std;
class Foo {
    public:
    void f() { cout << "a"; };
    virtual void g() = 0;
    virtual void c() = 0;
};
class Bar : public Foo
{
    public:
    void f() { cout << "b"; };
    virtual void g() { cout << "c"; };
    void c() { cout << "d"; };
    virtual void h() { cout << "e"; };
};
class Baz : public Bar
{
```

```
    public:
    void f() { cout << "f"; };
    virtual void g() { cout << "g"; };
    void c() { cout << "h"; };
    void h() { cout << "i"; };
};
class Qux : public Baz
{
    public:
    void f() { cout << "j"; };
    void h() { cout << "k"; };
};
int main() {
    Bar bar; bar.g();
    Baz baz; baz.h();
    Qux qux; qux.g();
    Foo &f1 = qux;
    f1.f(); f1.g(); f1.c();
    Bar &b1 = qux;
    b1.f(); b1.c(); b1.h();
    Baz &b2 = qux;
    b2.f(); b2.c(); b2.h();
    return 0;
}
```

Answer:

```
int main() {
Bar bar; bar.g(); // c
Baz baz; baz.h(); // i
Qux qux; qux.g(); // g
Foo &f1 = qux;
f1.f(); f1.g(); f1.c(); // a g h
Bar &b1 = qux;
b1.f(); b1.c(); b1.h(); // b h k
Baz &b2 = qux;
b2.f(); b2.c(); b2.h(); // f h k
return 0;
}
```