Basics: From C to C++

Computer Programming for Engineers (DSAF003-42)
Fall, 2021

Practice 9 : Polymorphism-II

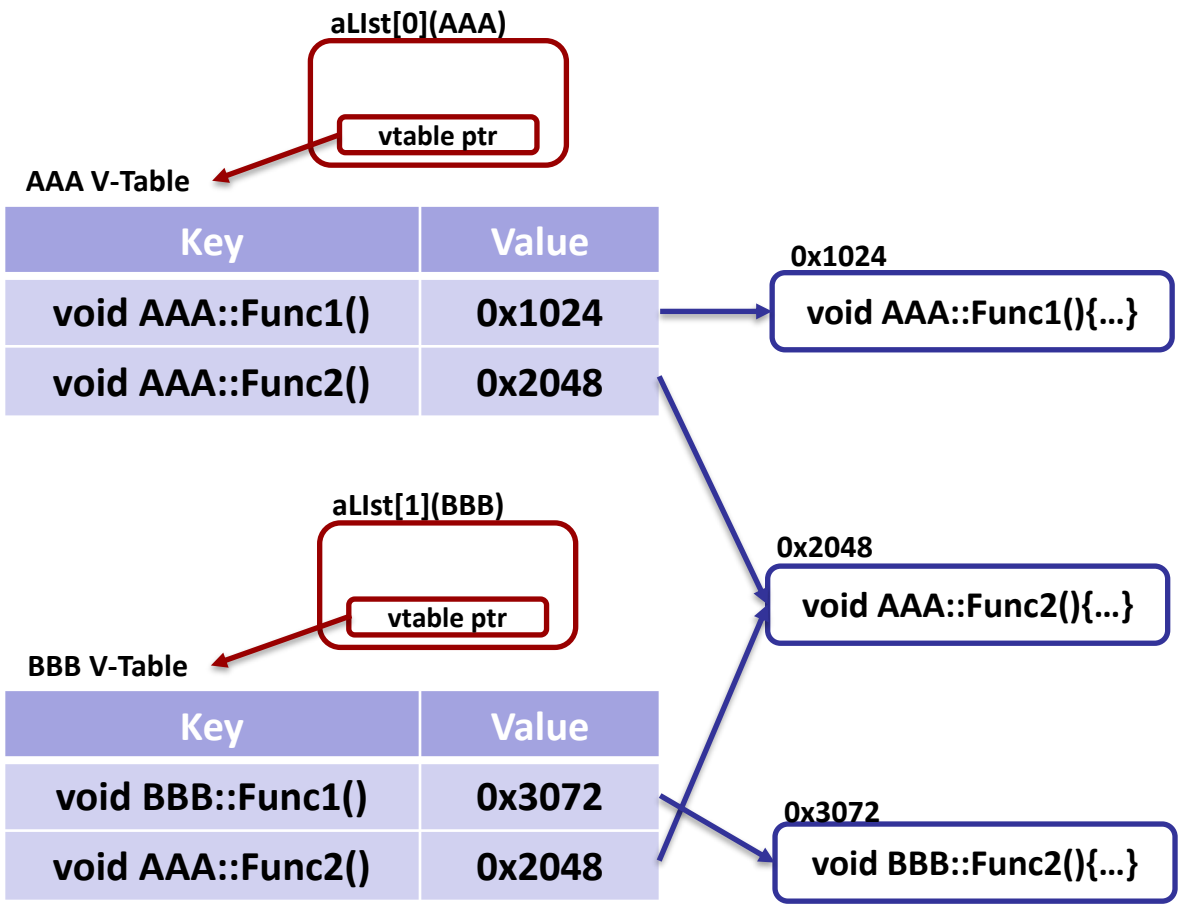
Instructor:

Youngjoong Ko (nlp.skku.edu)

Virtual Function Example

```cpp
#include <iostream>
using namespace std;
class AAA{
public:
    virtual void Func1(){
        cout<<"AAA::Func1"<<endl;
    }
    void Func2(){
        cout<<"AAA::Func2"<<endl;
    }
};
class BBB: public AAA{
public:
    virtual void Func1(){
        cout<<"BBB::Func1"<<endl;
    }
    void Fun2(){
        cout<<"BBB::Func2"<<endl;
    }
};
int main(){
    AAA* aList[2];
    aList[0] = new AAA();
    aList[1] = new BBB();
    for(int i=0; i<2; i++){
        aList[i]->Func1();
    }
    for(int i=0; i<2; i++){
        aList[i]->Func2();
    }
    return 0;
}
```

```
AAA::Func1
BBB::Func1
AAA::Func2
AAA::Func2
```



Override keyword Example

- **Override specifier ensures that the function is virtual and is overriding a virtual function from a base class.**

```cpp
#include <iostream>
using namespace std;
class AAA{
public:
    virtual void Func1(){
        cout<<"AAA::Func1"<<endl;
    }
    void Func2(){
        cout<<"AAA::Func2"<<endl;
    }
};
class BBB: public AAA{
public:
    virtual void Func1(){
        cout<<"BBB::Func1"<<endl;
    }
    void Fun2(){
        cout<<"BBB::Func2"<<endl;
    }
};
int main(){
    AAA* aList[2];
    aList[0] = new AAA();
    aList[1] = new BBB();
    for(int i=0; i<2; i++){
        aList[i]->Func1();
    }
    for(int i=0; i<2; i++){
        aList[i]->Func2();
    }
    return 0;
}
```

```
AAA::Func1
BBB::Func1
AAA::Func2
AAA::Func2
```

```cpp
#include <iostream>
using namespace std;
class AAA{
public:
    virtual void Func1(){
        cout<<"AAA::Func1"<<endl;
    }
    void Func2(){
        cout<<"AAA::Func2"<<endl;
    }
};
class BBB: public AAA{
public:
    void Func1() override{
        cout<<"BBB::Func1"<<endl;
    }
    void Fun2(){
        cout<<"BBB::Func2"<<endl;
    }
};
int main(){
    AAA* aList[2];
    aList[0] = new AAA();
    aList[1] = new BBB();
    for(int i=0; i<2; i++){
        aList[i]->Func1();
    }
    for(int i=0; i<2; i++){
        aList[i]->Func2();
    }
    return 0;
}
```

```
AAA::Func1
BBB::Func1
AAA::Func2
AAA::Func2
```

Final keyword Example

- **Final specifier ensures that the function is virtual and specifies that it may not be overridden by derived classes**

```
1   #include <iostream>
2   using namespace std;
3   class AAA{
4   public:
5       virtual void Func1() final{
6           cout<<"AAA::Func1"<<endl;
7       }
8       void Func2(){
9           cout<<"AAA::Func2"<<endl;
10      }
11  };
12  class BBB: public AAA{
13  public:
14      void Func1() {
15          cout<<"BBB::Func1"<<endl;
16      }
17      void Fun2(){
18          cout<<"BBB::Func2"<<endl;
19      }
20  };
21  int main(){
22      AAA* aList[2];
23      aList[0] = new AAA();
24      aList[1] = new BBB();
25      for(int i=0; i<2; i++){
26          aList[i]->Func1();
27      }
28      for(int i=0; i<2; i++){
29          aList[i]->Func2();
30      }
31      return 0;
32  }
```

- **If eliminate line 14 to 16, the output is**

```
AAA::Func1
AAA::Func1
AAA::Func2
AAA::Func2
```

```
/home/taehun/projects/lab8.cpp:14:10: error: virtual function ‘virtual void BBB::Func1()’ overriding final function
   14 |     void Func1() {
      |          ^~~~~
/home/taehun/projects/lab8.cpp:5:18: note: overridden function is ‘virtual void AAA::Func1()’
    5 |     virtual void Func1() final{
      |                  ^~~~~

Build finished with error(s).
```

Pure virtual function

- **A class containing pure virtual functions is an abstract class**
  - Assign NULL (0) value to virtual function instead of implementation

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

class Person{
public:
    Person() {};
    virtual ~Person() {};
    virtual void Action()=0; //Pure Virtual Functions
};
class Student : public Person{
public:
    Student() {};
    ~Student() {};
    void Action()   {
        cout << "Student" << endl;
    }
};
```

```cpp
class Professor : public Person{
public:
    Professor() {};
    ~Professor() {};
    void Action(){
        cout << "Professor" << endl;
    }
};
int main(){
    Student* student = new Student();
    Professor professor;

    student->Action();
    professor.Action();

    delete student;
    return 0;
}
```

```
Student
Professor
```

Exercise 1

- **Define Dog, smallDog, bigDog class**
  - Dog is derived class of Animal
  - SmallDog and bigDog is derived class of Dog

```cpp
#include <iostream>
#include <string>
using namespace std;
class Animal{
protected:
    string name;
public:
    Animal(string myname):name(myname){}
    virtual ~Animal(){}
    virtual void sound() const=0;
    virtual void info() const=0;
};
```

```cpp
class AnimalList{
private:
    Animal* animalList[10];
    int numAnimal=0;
public:
    ~AnimalList(){
        for(int i=0; i<numAnimal; i++){
            delete animalList[i];
        }
    }
    void addAnimal(Animal* pet){
        animalList[numAnimal++]=pet;
    }
    void sound() const{
        for(int i=0; i<numAnimal; i++){
            animalList[i]->sound();
        }
    }
    void info() const{
        for(int i=0; i<numAnimal; i++){
            animalList[i]->info();
        }
    }
};
```

```cpp
int main(){
    AnimalList aList;
    aList.addAnimal(new Dog("초코","brown"));
    aList.addAnimal(new smallDog("초코","brown",3));
    aList.addAnimal(new bigDog("초코","brown",10));

    aList.sound();
    aList.info();
    return 0;
}
```

```
Bark!
Bark!
Bark!
The brown dog's name is 초코
The small brown dog's name is 초코
The big brown dog's name is 초코
~Dog()
~smallDog()
~Dog()
~bigDog()
~Dog()
```

static_cast Example

- All types of conversions that are well-defined and allowed by the compiler can be performed using static_cast

```cpp
#include <iostream>
#include <string>
using namespace std;
class Animal {
public:
    virtual ~Animal(){}
    virtual void sound() = 0;
};
class Dog : public Animal{
private:
    string name;
public:
    Dog(string myname): name(myname){}
    void sound() override{
        cout<<"Bark!"<<endl;
    }
    void only_dog(){
        cout<<"dogs"<<endl;
    }
};
```

```cpp
class Cat : public Animal {
private:
    string name;
public:
    Cat(string myname): name(myname){}
    void sound() override{
        cout<<"Meow~"<<endl;
    }
    void only_cat() {
        cout<<"cats"<<endl;
    }
};
int main() {
    Animal* ani = new Cat("나비");

    Cat* cat = static_cast<Cat*>(ani);
    cat->sound();
    cat->only_cat();

    Dog* dog = static_cast<Dog*>(ani);
    dog->sound();
    dog->only_dog();

    return 0;
}
```

```
Meow~
cats
Meow~
dogs
```

dynamic_cast Example 1

- **dynamic_cast operator performs type-safe downcasts**
  - A downcast is the conversion of a pointer or reference to a class A to a pointer or reference to a class B, where class A is a base class of B

```cpp
#include <iostream>
using namespace std;
class Blog
{
public:
    Blog(){
        cout<<"Blog()"<<endl;
    };
    virtual ~Blog(){
        cout<< "~Blog()"<<endl;
    };
    virtual void Show(){
        cout << "This is Blog Class"<<endl;
    }
};
class Tistory : public Blog
{
public:
    Tistory(){
        cout<<"Tistory()"<<endl;
    };
    ~Tistory() override{
        cout << "~Tistory()"<<endl;
    };
    void Show() override final{
        cout << "This is Tistory Class"<<endl;
    }
};
```

```cpp
int main()
{
    // Blog* pBlog = new Blog();
    Blog* pBlog = new Tistory();
    pBlog->Show();

    Tistory* pTistory = dynamic_cast<Tistory*>(pBlog);
    if (pTistory == nullptr){
        cout << "Runtime Error"<<endl;
    }
    else{
        pTistory->Show();
    }
    delete pBlog;
    return 0;
}
```

①

```
Blog()
This is Blog Class
Runtime Error
~Blog()
```

②

```
Blog()
Tistory()
This is Tistory Class
This is Tistory Class
~Tistory()
~Blog()
```

dynamic_cast Example 2

```
#include <iostream>
#include <string>
using namespace std;
class Animal {
public:
    virtual ~Animal(){}
    virtual void sound() = 0;
};
class Dog : public Animal{
private:
    string name;
public:
    Dog(string myname): name(myname){}
    void sound() override{
        cout<<"Bark!"<<endl;
    }
    void only_dog(){
        cout<<"dogs"<<endl;
    }
};
class Cat : public Animal {
private:
    string name;
public:
    Cat(string myname): name(myname){}
    void sound() override{
        cout<<"Meow~"<<endl;
    }
    void only_cat() {
        cout<<"cats"<<endl;
    }
};
```

```
int main() {
    Animal* aList[2];
    aList[0] = new Cat("나비");
    aList[1] = new Dog("멍멍이");
    Cat* cat; Dog* dog;
    for(int i=0; i<2; i++){
        if(cat = dynamic_cast<Cat*>(aList[i])){
            cat->only_cat();
        }
        else{
            dog = dynamic_cast<Dog*>(aList[i]);
            dog->only_dog();
        }
    }
    delete aList[0];
    delete aList[1];
    return 0;
}
```

```
cats
dogs
```

Assignment

- **Using previous codes define Dog, Cat, AnimalList class**
  - The function all_animal() must use dynamic_cast

```
#include <iostream>
#include <string>
using namespace std;
class Animal{
protected:
    string name;
public:
    Animal(string myname):name(myname){}
    virtual ~Animal(){}
    virtual void sound() const=0;
    virtual void info() const=0;
};
```

```
int main(){
    AnimalList aList;
    aList.addAnimal(new Dog("초코","brown"));
    aList.addAnimal(new Cat("하양","white"));
    aList.addAnimal(new smallDog("초코","brown",3));
    aList.addAnimal(new bigDog("초코","brown",10));

    aList.all_animal();

    return 0;
}
```

```
dogs
The brown dog's name is 초코
cats
The white dog's name is 하양
dogs
The small brown dog's name is 초코
dogs
The big brown dog's name is 초코
~Dog()
~Cat()
~smallDog()
~Dog()
~bigDog()
~Dog()
```