Basics: From C to C++

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

Practice 8 : Polymorphism-I

Instructor:

Youngjoong Ko (nlp.skku.edu)

Overriding function Example

- **Decision of call function based on pointer type**

```cpp
#include <iostream>
using namespace std;
class Animal{
public:
    void MyFunc(){cout << "Animal" << endl;}
};
class Dog : public Animal{
public:
    void MyFunc(){cout << "Dog" << endl;}
};
class Cat : public Animal{
public:
    void MyFunc(){cout << "Cat" << endl;}
};
int main(){
    Animal* animals[10];
    animals[0] = new Animal();
    animals[1] = new Dog();
    animals[2] = new Cat();

    animals[0]->MyFunc();
    animals[1]->MyFunc();
    animals[2]->MyFunc();

    return 0;
}
```

```
Animal
Animal
Animal
```

Virtual function Example

- **Decision of call function based on the object pointed to by the pointer**

```cpp
#include <iostream>
using namespace std;
class Animal{
public:
    void MyFunc(){cout << "Animal" << endl;}
};
class Dog : public Animal{
public:
    void MyFunc(){cout << "Dog" << endl;}
};
class Cat : public Animal{
public:
    void MyFunc(){cout << "Cat" << endl;}
};
int main(){
    Animal* animals[10];
    animals[0] = new Animal();
    animals[1] = new Dog();
    animals[2] = new Cat();

    animals[0]->MyFunc();
    animals[1]->MyFunc();
    animals[2]->MyFunc();

    return 0;
}
```

```
Animal
Animal
Animal
```

```cpp
#include <iostream>
using namespace std;
class Animal{
public:
    virtual void MyFunc(){cout << "Animal" << endl;}
};
class Dog : public Animal{
public:
    virtual void MyFunc(){cout << "Dog" << endl;}
};
class Cat : public Animal{
public:
    void MyFunc(){cout << "Cat" << endl;}
};
int main(){
    Animal* animals[10];
    animals[0] = new Animal();
    animals[1] = new Dog();
    animals[2] = new Cat();

    animals[0]->MyFunc();
    animals[1]->MyFunc();
    animals[2]->MyFunc();

    return 0;
}
```

```
Animal
Dog
Cat
```

Inheritance Example without virtual

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

class Person{
private:
    string* name = new string;
    // string name;
public:
    Person(string myname){
        cout << "Person(string)"<<endl;
        *name = myname;
    }
    ~Person(){
        cout << "~Person()" << endl;
        delete name;
    }
    void show(){
        cout << "My name is " << *name <<endl;
    }
    string getName(){
        return *name;
    }
};
```

```cpp
class Student : public Person{
private:
    int* id = new int;
public:
    Student(string myname, int myid):Person(myname){
        cout << "Student(string, int)"<<endl;
        *id = myid;
    }
    ~Student(){
        cout << "~Student()" << endl;
        delete id;
    }
    void show(){
        cout << "My name is " << getName();
        cout << "and ID is "<< *id <<endl;
    }
};
int main(){
    Person* a = new Student("태훈",2021711919);
    Person* b = new Student("준희",2021711910);
    a->show();
    b->show();
    delete a;
    delete b;

    return 0;
}
```

```
Person(string)
Student(string, int)
Person(string)
Student(string, int)
My name is 태훈
My name is 준희
~Person()
~Person()
```

Inheritance Example with virtual

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

class Person{
private:
    string* name = new string;
    // string name;
public:
    Person(string myname){
        cout << "Person(string)"<<endl;
        *name = myname;
    }
    virtual ~Person(){
        cout << "~Person()" << endl;
        delete name;
    }
    virtual void show(){
        cout << "My name is " << *name <<endl;
    }
    string getName(){
        return *name;
    }
};
```

```cpp
class Student : public Person{
private:
    int* id = new int;
public:
    Student(string myname, int myid):Person(myname){
        cout << "Student(string, int)"<<endl;
        *id = myid;
    }
    ~Student(){
        cout << "~Student()" << endl;
        delete id;
    }
    void show(){
        cout << "My name is " << getName();
        cout << " and ID is "<< *id <<endl;
    }
};
int main(){
    Person* a = new Student("태훈",2021711919);
    Person* b = new Student("준희",2021711910);
    a->show();
    b->show();
    delete a;
    delete b;

    return 0;
}
```

```
Person(string)
Student(string, int)
Person(string)
Student(string, int)
My name is 태훈 and ID is 2021711919
My name is 준희 and ID is 2021711910
~Student()
~Person()
~Student()
~Person()
```

Exercise 1

- **Define Animal class, Dog class**
  - Dog class is a derived class of Animal
  - Define constructor, destructor, show function etc.

```
int main(){
    Animal* animals[2];
    animals[0] = new Dog("초코",10);
    animals[1] = new Dog("쿠키",6);
    animals[0]->Show();
    animals[1]->Show();

    delete animals[1];
    delete animals[0];

    return 0;
}
```

```
1th Animal()
1th Dog()
2th Animal()
2th Dog()
name is 초코 and age is 10
name is 쿠키 and age is 6
2th ~Dog()
2th ~Animal()
1th ~Dog()
1th ~Animal()
```

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 2

- **Define MainPlayer, SubPlayer, StarPlayer class Using previous Assignment**
- **Define PlayerList without any list of derived class of Player**
  - Must deallocate dynamic memory

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

class Player{
private:
    string name;
public:
    Player(string myname):name(myname){}
    string getname() const {
        return name;
    }
    virtual ~Player(){
        cout<<"~Player()"<<endl;
    }
    virtual void showinfo() const=0;
    virtual int getpay() const=0;
};
```

```
int main(){
    PlayerList players;
    players.addplayer(new MainPlayer("태훈",200));

    SubPlayer* b = new SubPlayer("춤원",15,5);
    b->addmatch(5);
    players.addplayer(b);

    StarPlayer* c = new StarPlayer("두영",500,30,10);
    c->addmatch(10);
    players.addplayer(c);

    players.addplayer(new MainPlayer("준희",300));

    players.showinfo();
    players.showtotal();

    return 0;
}
```

```
태훈's salary: 200
춤원's salary: 100
두영's salary: 900
준희's salary: 300
Total Salary: 1500
~MainPlayer()
~Player()
~SubPlayer()
~Player()
~StarPlayer()
~MainPlayer()
~Player()
~MainPlayer()
~Player()
~PlayerList
```