Virtual functions in C++

Όταν χρησιμοποιούμε κληρονομικότητα, είναι πολύ πιθανό η parent/base κλάση και η child/derived κλάση να έχουν συναρτήσεις-μέλη με το ίδιο όνομα, αφού κάθε child κλάση μπορεί να χρειάζεται να επεκτείνει τις συναρτήσεις της parent κλάσης.

Επίσης πολύ πιθανό είναι να δηλώσουμε δείκτες της parent κλάσης, οι οποίοι όμως τελικά θα χρησιμοποιηθούν για να δείχνουν σε αντικείμενα της child κλάσης. Αυτό γίνεται, αφού η child κλάση είναι ΚΑΙ parent κλάση, λόγω κληρονομικότητας.

Αυτό χρησιμεύει γιατί μπορεί να μην ξέρουμε από πριν τι είδος αντικειμένου θα δεσμεύσουμε στον δείκτη (αντικείμενο parent ή μίας από τις child κλάσεις), οπότε χρειαζόμαστε ένα δείκτη γενικού τύπου.

Με άλλα λόγια κάνουμε το εξής:

```
class A {...};
class B : public A {...};

A *A_ptr;
B b;
A_ptr = &b;
```

Ο δείκτης A_ptr μπορεί να δείξει τόσο σε αντικείμενα Α, όσο και σε αντικείμενα Β.

Το πρόβλημα που δημιουργείται είναι το εξής: αν και οι δύο κλάσεις έχουν μία συνάρτηση με το ίδιο όνομα, έστω common_function(), τότε η κλήση της συνάρτησης ως A_ptr->common_function() δεν θα έχει το επιθυμητό αποτέλεσμα.

Με άλλα λόγια, ενώ θα θέλαμε αφού ο δείκτης A_ptr δείχνει σε αντικείμενο Β, να κληθεί η συνάρτηση common_function της κλάσης Β, αυτό δεν συμβαίνει. Αντίθετα, καλείται η συνάρτηση common_function της κλάσης Α.

Αυτό συμβαίνει γιατί ο compiler δεν μπορεί να γνωρίζει σε τι αντικείμενο δείχνει ο A_ptr για να ξέρει ποιάς κλάσης την συνάρτηση θα καλέσει. Γι' αυτό θα καλέσει την συνάρτηση του τύπου στον οποίο δείχνει ο A_ptr, δηλαδή την common _function της Α.

Παράδειγμα

```
class Vehicle{
        public:
            void behavior()
            {
                cout << "I am a vehicle!" << endl;
            }
        };

class Car: public Vehicle{
        public:
            void behavior()
            {
                cout << "I am a car!" << endl;
            }
        };
```

```
int main()
{
   Vehicle a_vehicle;
   Car a_car;
   Vehicle   *vehicle_p;

   cout << "Vehicle behavior is: ";
   a_vehicle.behavior();

   cout << "Car behavior is: ";
   a_car.behavior();

   vehicle_p = &a_vehicle;
   cout << "Vehicle behavior is: ";
   vehicle_p->behavior();

   vehicle_p = &a_car;
   cout << "Car behavior is: ";
   vehicle_p->behavior();

   return 0;
}
```

Αποτέλεσμα

Vehicle behavior is: I am a vehicle!
Car behavior is: I am a car!
Vehicle behavior is: I am a vehicle!
Car behavior is: I am a vehicle!

Virtual συναρτήσεις

Για να λύσουμε αυτό το πρόβλημα, δηλώνουμε την συνάρτηση της parent κλάσης ως virtual.

Έτσι καλείται η συνάρτηση της child κλάσης ακόμη και αν ο δείκτης που δείχνει στο αντικείμενο της child κλάσης είναι τύπου parent κλάσης.

Δηλαδή *καλείται η κατάλληλη συνάρτηση ανάλογα με το τι τύπου είναι το αντικείμενο στο οποίο δείχνει ο δείκτης* και όχι ανάλογα με τι τύπου είναι ο ίδιος ο δείκτης.

Αν ωστόσο θέλουμε να καλέσουμε την virtual συνάρτηση της parent κλάσης μέσα στην συνάρτηση της child κλάσης, αυτό είναι δυνατόν με χρήση του τελεστή "::". Δηλαδή θα την καλέσουμε ως Parent::common_function().

Στην περίπτωση που μια συνάρτηση έχει οριστεί virtual στην parent κλάση, αλλά δεν έχει υλοποιηθεί στην child κλάση, καλείται αυτόματα η συνάρτηση της parent κλάσης και για το αντικείμενο της child κλάσης.

Παράδειγμα

```
class Vehicle{
        public:
            virtual void behavior()
            {
                cout << "I am a vehicle!" << endl;
            }
        };

class Car: public Vehicle{
        public:
            void behavior()
            {
                cout << "I am a car!" << endl;
            }
        };

class Motorcycle: public Vehicle{
        public:
            void behavior()
            {
                cout << "I am a motorcycle! " ;
                Vehicle::behavior();
            }
        };

class Truck: public Vehicle{   };
```

Δεν έχει οριστεί συνάρτηση behavior για την κλάση Truck!

```
int main()
{

   Vehicle a_vehicle;
   Car a_car;
   Motorcycle a_motorcycle;
   Vehicle *vehicle_p;

   vehicle_p = &a_vehicle;
   cout << "Vehicle behavior is: ";
   vehicle_p->behavior();

   vehicle_p = &a_car;
   cout << "Car behavior is: ";
   vehicle_p->behavior();

   vehicle_p = &a_motorcycle;
   cout << "Motorcycle behavior is: ";
   vehicle_p->behavior();

   vehicle_p = &a_truck;
   cout << "Truck behavior is: ";
   vehicle_p->behavior();

   return 0;
}
```

Αποτέλεσμα

Vehicle behavior is: I am a vehicle!
Car behavior is: I am a car!
Motorcycle behavior is: I am a motorcycle! I am a vehicle!
Truck behavior is: I am a vehicle!

Pure virtual συναρτήσεις

Οι συναρτήσεις της parent κλάσης μπορούν επίσης να δηλωθούν και ως pure virtual:

```
virtual void common _function() = 0;
```

Μία pure virtual συνάρτηση συνήθως *δεν έχει υλοποίηση*, χωρίς αυτό να είναι υποχρεωτικό.

Δεν μπορούν να οριστούν αντικείμενα μίας κλάσης, αν αυτή περιέχει έστω και μία pure virtual συνάρτηση. Μία τέτοια κλάση λέγεται *αφηρημένη/abstract*.

Η pure virtual συνάρτηση ΠΡΕΠΕΙ να υλοποιηθεί στην child κλάση, αφού διαφορετικά δεν θα μπορούν να οριστούν αντικείμενά της.

Συνήθως μία συνάρτηση θα οριστεί pure virtual αν δεν έχει κάποια χρήση ή κάποιο νόημα στην parent κλάση, αλλά πρέπει να υπάρχει στη child κλάση.

Μια pure virtual συνάρτηση μπορεί να υλοποιηθεί στη parent κλάση, είτε για να χρησιμοποιηθεί κάπου αλλού, είτε για να δοθεί μια default συμπεριφορά στις child κλάσεις (η συνάρτηση της child κλάσης μπορεί να καλέσει στην υποχρεωτική υλοποίησή της την pure virtual συνάρτηση της parent κλάσης). Επίσης, ένας βασικός λόγος για να υλοποιηθεί μία pure virtual συνάρτηση είναι ότι αυτή είναι στην εμβέλεια της αφηρημένης parent κλάσης κι έτσι έχουμε πρόσβαση στο private μέρος της, πράγμα που δεν έχουν τα παιδιά της κλάσης που την υλοποιούν.

Παράδειγμα

```
class Vehicle{
        public:
            virtual void behavior() = 0;

        };

class Car: public Vehicle{
        public:
            void behavior()
            {
                cout << "I am a car!" << endl;
            }
        };
```

```
int main()
{

    Car a_car;
    Vehicle *vehicle_p;

    vehicle_p = &a_car;
    cout << "Car behavior is: ";
    vehicle_p->behavior();

    return 0;
}
```

Αποτέλεσμα

```
Car behavior is: I am a car!
```

Παράδειγμα

```
class Vehicle{
        public:
            virtual void behavior() = 0;

        };

class Car: public Vehicle{
        public:
            void behavior()
            {
                cout << "I am a car!" << endl;
                Vehicle::behavior();
            }
        };

void Vehicle::behavior()
{
    cout << "I am a vehicle!" << endl;
}
```

```
int main()
{

    Car a_car;
    Vehicle *vehicle_p;

    vehicle_p = &a_car;
    cout << "Car behavior is: ";
    vehicle_p->behavior();

    return 0;
}
```

Αποτέλεσμα

```
Car behavior is: I am a car!
I am a vehicle!
```

Παράδειγμα

```
class Vehicle{
        public:
            virtual void behavior() = 0;

        };

class Car: public Vehicle{
        public:
            void behavior()
            {
                cout << "I am a car!" << endl;
            }
        };
```

```
int main()
{
   Vehicle a_vehicle;
   Car a_car;
   Vehicle *vehicle_p;

   vehicle_p = &a_car;
   cout << "Car behavior is: ";
   vehicle_p->behavior();

   return 0;
}
```

Δεν μπορούμε να έχουμε αντικείμενα Vehicle, γιατί η behavior είναι pure virtual!

Αποτέλεσμα

```
purevirtual.cpp: In function 'int main()':
purevirtual.cpp:20: error: cannot declare variable 'a_vehicle' to be of abstract type 'Vehicle'
purevirtual.cpp:4: note:   because the following virtual functions are pure within 'Vehicle':
purevirtual.cpp:6: note:   virtual void Vehicle::behavior()
```

Παράδειγμα

```
 public:
                virtual void behavior() = 0;

            };

class Car: public Vehicle{

            };

void Vehicle::behavior()
{
    cout << "I am a vehicle!" << endl;


}
```

Δεν υπάρχει υλοποίηση της behavior στην child κλάση!

```
int main()
{
   Car a_car;
   Vehicle *vehicle_p;

   vehicle_p = &a_car;
   cout << "Car behavior is: ";
   vehicle_p->behavior();

   return 0;
}
```

Αποτέλεσμα

```
purevirtual.cpp: In function 'int main()':
purevirtual.cpp:24: error: cannot declare variable 'a_car' to be of abstract type 'Car'
purevirtual.cpp:10: note:   because the following virtual functions are pure within 'Car':
purevirtual.cpp:17: note:  virtual void Vehicle::behavior()
```

Destructors

Οι destructors των parent κλάσεων πρέπει πάντα να δηλώνονται ως virtual, διαφορετικά δεν θα καλούνται οι destructors των child κλάσεων από τους δείκτες τύπου parent και η αποδέσμευση δεν θα γίνεται σωστά.

Παράδειγμα

```
class Vehicle{
        public:
            Vehicle()
            {
                cout << "I am a Vehicle!" << endl;
            }
             ~Vehicle()
            {
                cout << "Vehicle destroyed!" << endl;
            }
        };
```

```
class Car: public Vehicle{
        public:
            Car()
            {
                cout << "I am a Car!" << endl;
            }
            ~Car()
            {
                cout << "Car destroyed!" << endl;
            }
        };

int main()
{

   Vehicle *vehicle_p =  new Car();
   delete vehicle_p;

   return 0;
}
```

Αποτέλεσμα

Δεν καλείται ο destructor της κλάσης Car!

```
I am a Vehicle!
I am a Car!
Vehicle destroyed!
```

Παράδειγμα

```
class Vehicle{
        public:
            Vehicle()
            {
                cout << "I am a Vehicle!" << endl;
            }
            virtual ~Vehicle()
            {
                cout << "Vehicle destroyed!" << endl;
            }
        };
```

```
class Car: public Vehicle{
        public:
            Car()
            {
                cout << "I am a Car!" << endl;
            }
            ~Car()
            {
                cout << "Car destroyed!" << endl;
            }
        };

int main()
{

   Vehicle *vehicle_p =  new Car();
   delete vehicle_p;

   return 0;
}
```

Αποτέλεσμα

I am a Vehicle!
I am a Car!
Car destroyed!
Vehicle destroyed!