Αντικειμενοστρεφής Προγραμματισμός

Εργαστήριο 7

Πολυμορφισμός (Polymorphism)

Δείκτες σε παραγόμενες κλάσεις

Οι δείκτες σε αντικείμενα μιας κλάσης και σε αντικείμενα κλάσεων που παράγονται από αυτήν σχετίζονται μεταξύ τους. Έτσι όταν ένας δείκτης διακηρυχτεί στον τύπο της βασικής κλάσης, μπορεί να χρησιμοποιηθεί για την προσπέλαση στοιχείων σε αντικείμενα των παραγόμενων κλάσεων.

Παράδειγμα 1

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

class Animal
{
  public:
    void Display() {
    cout << "Base Animal Display() \n"; }
};

class Cat : public Animal
{
  public:
    void Display(){
    cout << "Derived Cat Display()\n";}
};

class Dog : public Animal
{
  public:
    void Display() {
    cout << "Derived Dog Display()\n"; }
};

class Fish : public Animal
{
  public:
  //no Display() provided
};

int main()
{
  Animal *Animal_ptr;
```

```
  Cat C;
  Dog D;
  Fish F;
  Animal_ptr = &C;
  Animal_ptr->Display();

  Animal_ptr = &D;
  Animal_ptr->Display();

  Animal_ptr = &F;
  Animal_ptr->Display();

  system("pause");
  return 0;
}
```

Παρατηρήστε ότι:

1. Είναι δυνατή η ανάθεση pointer τύπου Cat σε pointer τύπου Animal (επιτρέπεται γιατί η Cat προέρχεται από την Animal)
2. Το αντίστροφο δεν είναι δυνατό, δηλαδή η ανάθεση pointer τύπου Animal σε pointer τύπου Cat
3. Σε κάθε περίπτωση, όλες οι κλήσεις της συνάρτησης Display( ) έχουν το ίδιο αποτέλεσμα: καλείται η συνάρτηση της κλάσης Animal.

Δυναμικές – Εικονικές (Virtual) Συναρτήσεις

Η C++ επιτυγχάνει τον πολυμορφισμό κατά την εκτέλεση ενός προγράμματος χρησιμοποιώντας παραγόμενες κλάσεις και δυναμικές συναρτήσεις, Οι δυναμικές συναρτήσεις διακηρύσσονται με τη λέξη κλειδί *virtual* στη βασική κλάση ενώ διαφορετικές εκδόσεις τους (χωρίς να είναι απαραίτητη η λέξη κλειδί *virtual*) διακηρύσσονται στις παραγόμενες κλάσεις. Οι τελευταίες μπορούμε να τις καλέσουμε χρησιμοποιώντας ένα δείκτη στον τύπο της βασικής κλάσης. Στην περίπτωση αυτή η C++ προσδιορίζει, κατά την εκτέλεση του προγράμματος, ποια συνάρτηση θα χρησιμοποιηθεί από τον τύπο του αντικειμένου του οποίου η διεύθυνση δόθηκε στο δείκτη.

Τα πρότυπα των δυναμικών συναρτήσεων πρέπει να ταυτίζονται κατά τη διακήρυξη τους τόσο στη βασική όσο και στις παραγόμενες κλάσεις, για να έχουμε τη δυνατότητα να συντάξουμε κώδικα γενικής μορφής τον οποίο να εκμεταλλεύονται διαφορετικού τύπου αντικείμενα. Επίσης οι δυναμικές συναρτήσεις πρέπει να είναι μέλη της βασικής και των παραγόμενων κλάσεων αλλά όχι φιλικές συναρτήσεις, χωρίς αυτό να τις αποκλείει να είναι φιλικές για άλλες κλάσεις.

Παράδειγμα 2

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

class Animal
{
  public:
    virtual void Display() {
    cout << "Base Animal Display() \n"; }
};

class Cat : public Animal
{
  public:
    void Display(){
    cout << "Derived Cat Display()\n";}
};

class Dog : public Animal
{
  public:
```

```
    void Display() {
    cout << "Derived Dog Display()\n"; }
};

class Fish : public Animal
{
  public:
  //no Display() provided
};

int main()
{
  Animal *Animal_ptr;
  Cat C;
  Dog D;
  Fish F;
  Animal_ptr = &C;
  Animal_ptr->Display();

  Animal_ptr = &D;
  Animal_ptr->Display();

  Animal_ptr = &F;
  Animal_ptr->Display();

  Animal_ptr = new Cat;
  Animal_ptr->Display();

  return 0;
}
```

Παρατηρήστε πως μπορούμε και δυναμικά να δημιουργήσουμε αντικείμενα με την εντολή new. Μπορείτε να επεκτείνετε τον προηγούμενο κώδικα ορίζοντας έναν πίνακα από pointers σε Animal και αναθέτοντας σε αυτόν διαδοχικά διάφορα αντικείμενα Cat, Dog και Fish.

Άσκηση 1
Στον παραπάνω κώδικα να προσθέσετε τις μεταβλητές age και name στην κλάση Animal. Επίσης, να προσθέσετε και μια virtual συνάρτηση `int isOld( )` η οποία θα επιστρέφει 1 ή 0 αν το ζώο είναι μεγάλο ή όχι αντίστοιχα.

Προφανώς για κάθε είδος τα όρια του γήρατος είναι διαφορετικά. Έστω ότι ένας σκύλος είναι γέρος όταν είναι πάνω από 10 έτη, μια γάτα πάνω από 15 και ένα ψάρι πάνω από 2 έτη.

Να υλοποιήσετε την isOld για κάθε υποκλάση, ώστε να επιστρέφει την σωστή πληροφορία. Επίσης, να τροποποιήσετε την Display ώστε να εκτυπώνει το όνομα και την ηλικία του ζώου.

Τέλος να τροποποιήσετε και την main ώστε να δείτε ότι οι κλάσεις σας λειτουργούν σωστά.

Καθαρά Δυναμικές – Εικονικές (Virtual) Συναρτήσεις – Abstract κλάσεις

Όπως διαπιστώσαμε στο προηγούμενο παράδειγμα όταν μια δυναμική συνάρτηση δεν ορίζεται για μια παραγόμενη κλάση, τότε στη θέση της θα εκτελεστεί η έκδοση που ορίζεται στη βασική κλάση. Σε πολλές όμως περιπτώσεις η συνάρτηση αυτή δεν εκτελεί καμιά εργασία ή απλώς μπορεί να εμφανίζει ένα μήνυμα. Επίσης υπάρχουν και οι περιπτώσεις που υπάρχει η βεβαιότητα ότι η συνάρτηση που θα εκτελέσει μια

παραγόμενη κλάση είναι αυτή που ορίζεται από την ίδια και όχι από τη βασική της. Η λύση σε αυτές τις περιπτώσεις είναι η καθαρά δυναμική συνάρτηση.

Η γενική μορφή της διακήρυξης μιας καθαρά δυναμικής συνάρτησης είναι:

virtual τύπος επιστρ. τιμής όνομα συνάρτησης (κατάλογος παραμέτρων) = 0;

Η καθαρά δυναμική συνάρτηση δεν έχει ορισμό σχετιζόμενο με αυτή. Επομένως κάθε παραγόμενη κλάση πρέπει να έχει τη δική της έκδοση για τη συνάρτηση, αφού δεν μπορεί να χρησιμοποιήσει της βασικής, αλλιώς ο μεταγλωττιστής θα δώσει μήνυμα λάθους.

Η κλάση που περιέχει τουλάχιστον μια καθαρά δυναμική συνάρτηση ονομάζεται abstract και έχει το ιδιαίτερο χαρακτηριστικό ότι δεν μπορούμε να διακηρύξουμε αντικείμενά της.

Παράδειγμα 3

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

//abstract class
class Shape{
  public:
    virtual float Area()=0;       //pure virtual functions
    virtual float Perimeter()=0;
};

class Square:public Shape{
  public:
    Square(float s=0){side=s;}
    float Area(){return side*side;}
    float Perimeter(){return 4*side;}
  private:
    float side;
};

class Rectangle:public Shape{
  private:
    float length,width;
  public:
    Rectangle(float l=0,float w=0){length=l;width=w;}
    float Area(){return length*width;}
    float Perimeter(){return 2*length+2*width;}
};

int main()
{
    Shape *s[3];
    s[0] = new Square(5.31);
    s[1] = new Rectangle(3, 7.9);

    cout<<"Area of square is " <<s[0]->Area() <<" square units.\n";
    cout <<"Perimeter of rectangle is " <<s[1]->Perimeter()<<"units.\n";
    system("pause");
    return 0;
}
```

Άσκηση 2:
Δοκιμάστε να δημιουργήσετε ένα αντικείμενο της κλάσης Shape. Επιτρέπεται και γιατί;

Άσκηση 3:
Συνεχίζοντας το παραπάνω πρόγραμμα, να υλοποιήσετε και μια κλάση Circle που θα είναι και αυτή υποκλάση της Shape.
Στην συνέχεια, να εισάγετε και ένα αντικείμενο τύπου Circle στον πίνακα s (στην θέση s[2]).
Τέλος, στην main, να υλοποιήσετε ένα loop το οποίο θα υπολογίζει το άθροισμα των περιμέτρων όλων των Shapes, καθώς και το μεγαλύτερο εμβαδό από όλα τα Shapes που βρίσκονται στον πίνακα s.

Χρήσεις του πολυμορφισμού

Ο πολυμορφισμός είναι μια πολύ χρήσιμη ιδιότητα και μέσω αυτού μπορούμε να πετύχουμε σημαντικά πράγματα:

1. Μπορούμε να τοποθετούμε αντικείμενα διαφορετικού τύπου (αλλά που έχουν ένα κοινό πρόγονο) σε ένα πίνακα ή ένα vector και να τα χειριστούμε από κοινού

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

class Animal
{
  public:
    virtual void Display() {
    cout << "Base Animal Display() \n"; }
};

class Cat : public Animal
{
  public:
    void Display(){
    cout << "Derived Cat Display()\n";}
};

class Dog : public Animal
{
  public:
    void Display() {
    cout << "Derived Dog Display()\n"; }
};

class Fish : public Animal
{
  public:
  //no Display() provided
};

int main()
{
  vector<Animal*> v;
  v.push_back(new Cat); // Cat object is dynamically created using "new"
  v.push_back(new Dog);
  v.push_back(new Fish);
  v.push_back(new Dog);

  for(int i=0; i<v.size(); i++)
        v[i]->Display(); //the function of each class will be called

  system("pause");
  return 0;
}
```

2. Μπορούμε να έχουμε συναρτήσεις που απευθύνονται σε περισσότερες από μια κλάσεις. Στο παρακάτω παράδειγμα έχουμε δύο συναρτήσεις (addShapeAreas και addShapePerimeters) οι οποίες μπορούν να πάρουν ως παραμέτρους οποιοδήποτε αντικείμενο προέρχεται από την κλάση Shape. Δηλαδή μπορουν να πάρουν ως παραμέτρους, είτε δύο Square, είτε δύο Rectangle, είτε από ένα αντικείμενο από κάθε κλάση.

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

```
//abstract class
class Shape{
  public:
    virtual float Area()=0;   //pure virtual functions
    virtual float Perimeter()=0;
};

class Square:public Shape{
  public:
    Square(float s=0){side=s;}
    float Area(){return side*side;}
    float Perimeter(){return 4*side;}
  private:
    float side;
};

class Rectangle:public Shape{
  private:
    float length,width;
  public:
    Rectangle(float l=0,float w=0){length=l;width=w;}
    float Area(){return length*width;}
    float Perimeter(){return 2*length+2*width;}
};

float addShapeAreas(Shape *s1, Shape *s2)
{
    return s1->Area() + s2->Area();
}

float addShapePerimeters(Shape *s1, Shape *s2)
{
    return s1->Perimeter() + s2->Perimeter();
}

int main()
{
    Square sq(5.31);
    Rectangle r(3, 7.9);

    cout << "Add shape areas: " << addShapeAreas(&sq, &r) << endl;
    cout << "Add shape perimeters: " << addShapePerimeters(&sq, &r) << endl;

    system("pause");
    return 0;
}
```

Άσκηση 4
Να επεκτείνετε την Άσκηση 1 ορίζοντας μέσα στην main έναν **πίνακα** από pointers σε Animal (μεγέθους 10).

Στην συνέχεια, στην main να υλοποιήσετε με loop ένα μενού μέσω του οποίου ο χρήστης θα επιλέγει τι είδος ζώου θα τοποθετήσει στον πίνακα κάθε φορά. Αν πχ. ο χρήστης επιλέξει σκύλο, τότε θα πρέπει να δημιουργήσετε (*δυναμικά με τον τελεστή new*) ένα νέο αντικείμενο σκύλου (new Dog) το οποίο θα πρέπει να τοποθετήσετε στην κατάλληλη θέση του πίνακα.

Όταν ο χρήστης τελειώσει με τις επιλογές του, με ένα άλλο loop να εκτυπώσετε στην οθόνη πληροφορίες για κάθε ένα ζώο που υπάρχει μέσα στον πίνακα (όνομα, ηλικία και αν είναι ηλικιωμένο ή όχι).

Παράρτημα - Παράδειγμα με window class hierarchy

Το πιο κλασσικό παράδειγμα χρήσης των virtual functions είναι κατά τον παραθυρικό προγραμματισμό. Οι διάφορες βιβλιοθήκες που μπορούμε να χρησιμοποιήσουμε για να δημιουργήσουμε παράθυρα στις εφαρμογές μας, έχουν ορίσει μια ιεραρχία κλάσεων που παριστάνουν τα διαφορετικά components (windows, panels, buttons, combo boxes κλπ.). Γενικώς όλες αυτές οι κλάσεις κληρονομούν από μια αρχική κλάση, ψηλά στην ιεραρχία (ας την πούμε Component).
Στο παρακάτω απλό παράδειγμα, η κλάση Component ορίζει μέσα της μόνο μια συνάρτηση, την **pure virtual** function paint(). Κάθε κλάση που κληρονομεί από αυτήν, θα πρέπει να την υλοποιήσει. Έτσι, οι κλάσεις CommandButton και TextBox υλοποιούν αυτή την συνάρτηση. Όταν κληθεί αυτή η συνάρτηση για κάποιο από αυτά τα αντικείμενα, θα πρέπει αυτό να ζωγραφίσει τον εαυτό του στην οθόνη.

Στην συνάρτηση main, δημιουργούμε ένα παράθυρο, το οποίο περιέχει δύο Components: ένα TextBox και ένα CommandButton. Παρατηρούμε ότι έχουμε ορίσει ένα πίνακα που περιέχει pointers σε Component (την μητρική κλάση), και ότι σε αυτό τον πίνακα αποθηκεύουμε τους δείκτες των δύο components. Έχουμε με αυτό τον τρόπο την δυνατότητα κοινού χειρισμού όλων των components του παραθύρου.

Αν μετακινήσουμε το παράθυρο αυτό, θα πρέπει να το ζωγραφίσουμε πάλι στην οθόνη. Αυτό γίνεται δίνοντας την εντολή paint σε κάθε ένα από τα components που βρίσκονται πάνω στο παράθυρο. Για να γίνει αυτό, αρκεί να κάνουμε ένα loop σε όλο τον πίνακα και να καλέσουμε την paint() για όλα τα στοιχεία του πίνακα.

```
#include "stdafx.h"
#include <iostream>

using namespace std;

class Component
{
public:
	virtual void paint() = 0;

};

class CommandButton : public Component
{
public:
	void paint()
	{
		cout<<"Derived class Command Button - Overridden C++ virtual function"
<< endl;
	}
};

class TextBox : public Component
{
public:
	void paint()
	{
		cout<<"Derived class TextBox - Overridden C++ virtual function" <<
endl;
	}
};


int main()
{
	Component *components[2];

	components[0] = new TextBox();
	components[1] = new CommandButton();

	for(int i=0;i<2;i++)
```

```
            components[i]->paint();

      system("pause");
      return 0;
}
```