Εικονικές συναρτήσεις και πολυμορφισμός

Υπερφόρτωση συναρτήσεων

Function overloading

- Η υπερφόρτωση συναρτήσεων είναι η δυνατότητα να ορίζουμε δύο ή περισσότερες συναρτήσεις με το ίδιο όνομα αλλά διαφορετικές παραμέτρους.
  - Συναντάται στις αντικειμενοστραφείς γλώσσες προγραμματισμού,
  - Αποτελεί μια μορφή πολυμορφισμού.
- Η χρήση του ίδιου ονόματος για συναρτήσεις που επιτελούν την ίδια λειτουργία σε διαφορετικά είδη δεδομένων αυξάνει την αναγνωσιμότητα του προγράμματος.
  - Παράδειγμα: Έστω ότι χρειαζόμαστε μια συνάρτηση που να προσθέτει δύο αριθμούς, μια συνάρτηση που να προσθέτει τρεις αριθμούς και μια συνάρτηση που να προσθέτει τέσσερεις αριθμούς.
  - Εξυπηρετεί να δώσουμε και στις τρεις συναρτήσεις το όνομα add.
  - Ο Compiler καταλαβαίνει ποια συνάρτηση θέλουμε να καλέσουμε κάθε φορά από τον αριθμό των ορισμάτων της κλήσης.
- Οι διαφορετικές εκδόσεις μιας υπερφορτωμένης συνάρτησης μπορεί να διαφέρουν είτε ως προς τον αριθμό των παραμέτρων είτε ως προς τον τύπο (ή και τα δύο).
- Στη C++, εάν ο Compiler δεν βρει ορισμό συνάρτησης που να ταιριάζει ακριβώς στα ορίσματα της κλήσης προσπαθεί να επιτύχει το ταίριασμα κάνοντας κατάλληλες αναβαθμίσεις ή μετατροπές δεδομένων (π.χ. από float σε double, από double σε int, κλπ), ανάλογες με αυτές που συμβαίνουν όταν γίνεται ανάθεση μιας τιμής σε μεταβλητή που έχει δηλωθεί με διαφορετικό τύπο δεδομένων.
- Εάν δεν μπορέσει, εμφανίζει μήνυμα σφάλματος.

Παράδειγμα

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

class A {
public:
   double f(double x) {
       cout << "double f(double)" << endl;
       return x/2.0;
   }
   double f(double x1, double x2) {
       cout << "double f(double x1,double x2)" << endl;
       return x1/x2;
   }
   int f(int i) {
       cout << "int f(int x1)" << endl;
       return i/2;
   }
};


int main()
{
   A a;

   cout << a.f(3.0) << endl;
   cout << a.f(5.0, 2.0) << endl;
   cout << a.f(3) << endl;
   cout << a.f(7,2) << endl;

   return 0;
}
```

Αποτέλεσμα εκτέλεσης του προγράμματος

```
double f(double)
1.5
double f(double x1,double x2)
2.5
int f(int x1)
1
double f(double x1,double x2)
3.5
```

Υπερίσχυση συναρτήσεων

Function overriding

- Υπερίσχυση συναρτήσεων έχουμε όταν η ίδια συνάρτηση-μέλος ορίζεται και στη βασική κλάση και σε κάποια παράγωγη κλάση.
  - Αποτελεί και αυτή μορφή πολυμορφισμού
- Όταν καλούμε τη συνάρτηση σε ένα στιγμιότυπο της παράγωγης κλάσης, η εκδοχή της συνάρτησης στην παράγωγη κλάση *υπερισχύει* της εκδοχής της στη βασική.
- Φυσικά, όταν η συνάρτηση καλείται σε στιγμιότυπο της βασικής κλάσης, χρησιμοποιείται ο ορισμός που υπάρχει μέσα στη βασική κλάση.
- Εάν θέλουμε να χρησιμοποιήσουμε τη συνάρτηση της βασικής κλάσης σε ένα στιγμιότυπο της παράγωγης θα πρέπει να χρησιμοποιήσουμε τη σύνταξη

  όνομα βασικής κλάσης::όνομα συνάρτησης
- Η ίδια σύνταξη χρησιμοποιείται και όταν θέλουμε να καλέσουμε τη συνάρτηση της βασικής κλάσης από τον κώδικα της παράγωγης.

Παράδειγμα

Έστω ότι η βασική και η παράγωγη κλάση έχουν οριστεί ως εξής:

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

class base {
public:
   double f(double x) {
       cout << "base::f" << endl;
       return x/2.0;
   }
};

class derived:public base {
public:
   double f(double x) {
       cout << "derived::f" << endl;
       return x/4;
   }
};
```

Η έξοδος του προγράμματος για διάφορες βασικές συναρτήσεις main

```
int main()
{
   derived a;
   cout << a.f(3.0) << endl;
   return 0;
}
```

```
derived::f
0.75
```

```
 int main()
 {
    base a;
    cout << a.f(3.0) << endl;
    return 0;
 }
```

```
base::f
1.5
```

```
  int main()
  {
     derived a;
     cout << a.base::f(3.0) << endl;
     return 0;
  }
```

```
base::f
1.5
```

Κλήση της συνάρτησης-μέλους της βασικής κλάσης από τον κώδικα της παράγωγης

```
class derived:public base {
public:
   double f(double x) {
      cout << "derived::f" << endl;
      return base::f(x);
   }
};
int main()
{
   derived a;
   cout << a.f(3.0) << endl;
   return 0;
}
```

```
derived::f
base::f
1.5
```

Προσοχή: Εάν αντί για return base::f(x) είχαμε βάλει απλώς return f(x), η συνάρτηση θα καλούσε αναδρομικά τον εαυτό της και το πρόγραμμα θα έπεφτε σε ατέρμονα βρόχο (endless loop).

- Τόσο στη βασική κλάση όσο και στην παράγωγη μπορούμε να έχουμε περισσότερες από μία εκδοχές της συνάρτησης (υπερφόρτωση).
- Εάν στην παράγωγη κλάση έχει οριστεί έστω και μια μορφή μιας συνάρτησης-μέλους της βασικής κλάσης, ο ορισμός αυτός υπερισχύει όλων των μορφών της συνάρτησης που έχουν οριστεί στη βασική κλάση.
  - Οι υπόλοιπες μορφές «κρύβονται» από τα στιγμιότυπα της παράγωγης κλάσης.
  - Όταν καλούμε τη συνάρτηση σε στιγμιότυπο της παράγωγης κλάσης, ο Compiler προσπαθεί να ταιριάξει τα ορίσματα της κλήσης (ενδεχομένως με μετατροπές τύπων δεδομένων) με τις μορφές της συνάρτησης που έχουν οριστεί στην παράγωγη κλάση.
  - Αν δεν το καταφέρει βγάζει μήνυμα σφάλματος, έστω κι αν τα ορίσματα ταιριάζουν σε κάποια μορφή που έχει οριστεί στη βασική κλάση και δεν έχει ξαναοριστεί στην παράγωγη.
- Φυσικά μπορούμε να καλέσουμε τις μορφές που έχουν οριστεί στη βασική κλάση χρησιμοποιώντας τον τελεστή ::

Παράδειγμα

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

class base {
public:
   double f(double x) {
       cout << "base::double with one parameter." << endl;
       return x/2.0;
   }
      double f(double x1, double x2) {
       cout << "base::double with two parameters." << endl;
       return x1/x2;
   }
};

class derived:public base {
public:

   double f(double x) {
       cout << "derived::double " << endl;
       return x/4;
   }
};
```

```
int main()
{
   derived a;

   cout << a.f(3.0) << endl;
//  cout << a.f(5.0, 2.0) << endl;
   cout << a.base::f(3.0,2.0) << endl;
   cout << a.base::f(5.0) << endl;
   return 0;
}
```

Αν δίναμε αυτή την εντολή, ο Compiler θα έβγαζε λάθος. Η εκδοχή της συνάρτησης με δύο παραμέτρους δεν είναι ορατή από την παράγωγη κλάση.

Ο σωστός τρόπος για να καλέσουμε τις συναρτήσεις της βασικής κλάσης

Το αποτέλεσμα της εκτέλεσης του προγράμματος

```
derived::double
0.75
base::double with two parameters.
1.5
base::double with one parameter.
2.5
```

Παράδειγμα

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

class base {
public:
   double f(double x) {
       cout << "double f(double)" << endl;
       return x/2.0;
   }
};

class derived:public base {
public:
   int f(int i) {
       cout << "int f(int x1)" << endl;
       return i/2;
   }
};
```

Δεν γίνεται υπερφόρτωση της συνάρτησης. Η εκδοχή της παράγωγης κλάσης με το ακέραιο όρισμα υπερισχύει της εκδοχής της βασικής κλάσης με το όρισμα κινητής υποδιαστολής.

```
int main()
{
   derived a;

   cout << a.f(3.0) << endl;
   cout << a.f(3) << endl;
   cout << a.base::f(10.0) << endl;
   cout <<a.base::f(5) << endl;

   return 0;
}
```

Δεν καλείται η εκδοχή της συνάρτησης με όρισμα κινητής υποδιαστολής. Καλείται η εκδοχή που έχει οριστεί στην παράγωγη κλάση με κατάλληλη μετατροπή του ορίσματος.

Κλήση των μορφών της βασικής κλάσης.

Αποτέλεσμα της εκτέλεσης του προγράμματος

```
int f(int x1)
1
int f(int x1)
1
double f(double)
5
double f(double)
2.5
```

Εικονικές συναρτήσεις

Virtual Functions

- Τα πράγματα γίνονται ακόμη πιο ενδιαφέροντα όταν αναφερόμαστε σε στιγμιότυπο της παράγωγης κλάσης μέσω δεικτών.
- Μπορούμε να χρησιμοποιήσουμε δείκτη είτε προς αντικείμενα της βασικής κλάσης είτε προς αντικείμενα της παράγωγης κλάσης.
  - Στην πρώτη περίπτωση καλείται η συνάρτηση της βασικής κλάσης ενώ στη δεύτερη η συνάρτηση της παράγωγης.
- Ο Compiler αποφασίζει ποια συνάρτηση θα κληθεί κάθε φορά, ανάλογα με τον τύπο του δείκτη και όχι ανάλογα με τον τύπο του συγκεκριμένου στιγμιοτύπου προς το οποίο δείχνει ο δείκτης.
- Ο τύπος του στιγμιοτύπου, προς το οποίο δείχνει ο δείκτης, γενικά δεν είναι γνωστός στον Compiler κατά τον χρόνο μετάφρασης του προγράμματος.

Παράδειγμα

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

class base {
public:
   void f() {
      cout << "base::f()" << endl;
   }
};

class derived:public base {
public:
   void f() {
      cout << "derived::f()" <<endl;
   }
};
```

```
int main()
{

    base *p_base;
    derived oderived, *p_derived;

    p_base = &oderived;
    p_derived = &oderived;
    p_base->f();
    p_derived->f();

    return 0;
}
```

Αποτέλεσμα εκτέλεσης του προγράμματος

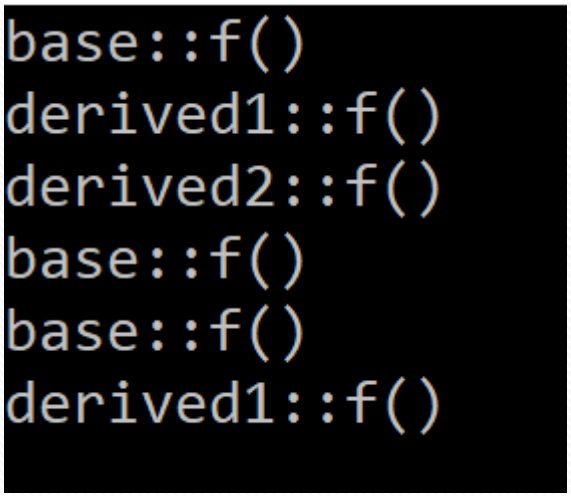
```
base::f()
derived::f()
```

- Υπάρχει και η δυνατότητα το ποια συνάρτηση θα κληθεί κάθε φορά να μην προσδιορίζεται εξαρχής από τον Compiler αλλά να αποφασίζεται δυναμικά, κατά το χρόνο εκτέλεσης του προγράμματος, ανάλογα με τον τύπο του στιγμιοτύπου προς το οποίο δείχνει ο δείκτης.
- Για να γίνει αυτό, αρκεί στον ορισμό της συνάρτησης στη βασική κλάση να προστεθεί ο χαρακτηρισμός “virtual”.
  - Οι συναρτήσεις που ορίζονται με τον τρόπο αυτόν ονομάζονται “εικονικές συναρτήσεις” (virtual functions).
- Ο μηχανισμός αυτός ονομάζεται “πολυμορφισμός χρόνου εκτέλεσης” (run-time polymorphism) και αποτελεί ένα από τα πιο ισχυρά χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού.
  - Ο ορισμός της εικονικής συνάρτησης στη βασική κλάση δηλώνει μια γενική ενέργεια που μπορεί να εφαρμοστεί σε διάφορα είδη συναφών αντικειμένων και ορίζει τη διεπαφή της συνάρτησης.
  - Ο επανορισμός της στις παράγωγες κλάσεις ορίζει πώς ακριβώς εκτελείται η γενική ενέργεια σε κάθε είδος αντικειμένου.
  - Το σημαντικό είναι ότι η σύνταξη του προγράμματος μπορεί να γίνεται με τρόπο γενικευμένο, χωρίς να καθορίζεται ακριβώς το είδος του αντικειμένου.
- Οι εικονικές συναρτήσεις μπορούν να χρησιμοποιηθούν και ως απλές (μη εικονικές) συναρτήσεις.
  - Τότε όμως χάνεται το πλεονέκτημα του γενικευμένου τρόπου σύνταξης του προγράμματος.
  - Ο προγραμματιστής πρέπει να δηλώνει ρητά ποια εκδοχή της συνάρτησης θα χρησιμοποιηθεί.

Παράδειγμα

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

class base {
public:
   virtual void f() {
       cout << "base::f()" << endl;
   }
};

class derived1:public base {
public:
   void f() {
       cout << "derived1::f()" <<endl;
   }
};

class derived2:public base {
public:
   void f() {
       cout << "derived2::f()" <<endl;
   }
};
```

```
int main()
{

   base b, *p;
   derived1 d1;
   derived2 d2;

   p = &b;
   p->f(); //καλείται η base::f()
   p = &d1;
   p->f(); //καλείται η derived1::f()
   p = &d2;
   p->f(); //καλείται η derived2::f()
   p->base::f(); //εάν θέλουμε να καλέσουμε την έκδοση της βασικής κλάσης
   b.f(); //Κλήση της base::f ως κανονικής (μη εικονικής) συνάρτησης
   d1.f(); //Κλήση της derived1::f ως κανονικής (μη εικονικής) συνάρτησης

   return 0;
}
```



- Ο χαρακτηρισμός virtual δεν χρειάζεται να επαναλαμβάνεται όταν ξαναορίζουμε τη συνάρτηση στις παράγωγες κλάσεις.
  - Δεν είναι όμως και λάθος να τον βάλουμε.
- Ο ορισμός της εικονικής συνάρτησης στις παράγωγες κλάσεις πρέπει να έχει τις ίδιες ακριβώς παραμέτρους όπως και στη βασική κλάση.
  - Διαφορετικά, ο εικονικός χαρακτήρας της συνάρτησης χάνεται και δεν εφαρμόζεται ο πολυμορφισμός χρόνου εκτέλεσης.

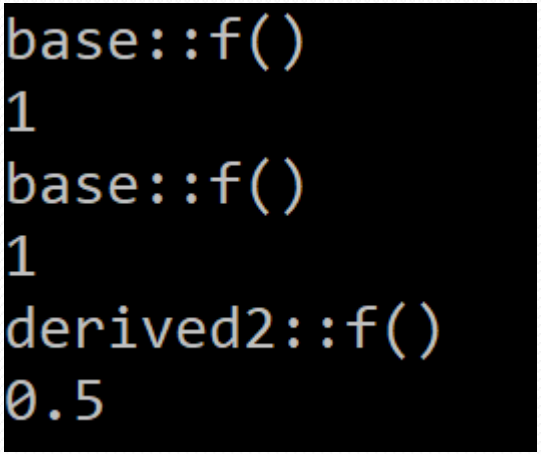
```
#include <iostream>
using namespace std;

class base {
public:
   virtual double f(double x) {
      cout << "base::f()" << endl;
      return x/2.0;
   }
};

class derived1:public base {
public:
   int f(int x) {
      cout << "derived1::f()" <<endl;
      return x/2;
   }
};

class derived2:public base {
public:
   double f(double x) {
      cout << "derived2::f()" <<endl;
      return x/4.0;
   }
};
```

Διαφορετικές παράμετροι. Δεν θεωρείται επανορισμός εικονικής συνάρτησης.

Ίδιες παράμετροι. Θεωρείται επανορισμός εικονικής συνάρτησης.

```
int main()
{

    base b, *p;
    derived1 d1;
    derived2 d2;

    p = &b;
    p->f(2); //καλείται η base::f()
    p = &d1;
    p->f(2); //καλείται η base::f()
    p = &d2;
    p->f(2); //καλείται η derived2::f()

    return 0;
}
```

Αποτελέσματα



Ο χαρακτηρισμός μιας συνάρτησης μέλους ως εικονικής κληρονομείται και στις παράγωγες κλάσεις.

```
class base {
public:
   virtual void f() {
      cout << "base::f()" << endl;
   }
};

class derived1:public base {
public:
   void f() {
      cout << "derived1::f()" <<endl;
   }
};

class derived2:public derived1 {
public:
   void f() {
      cout << "derived2::f()" <<endl;
   }
};
```

Θεωρείται και αυτή εικονική συνάρτηση.

```
int main()
{

    base b, *p;
    derived1 d1;
    derived2 d2;

    p = &b;
    p->f(); //καλείται η base::f()
    p = &d1;
    p->f();  //καλείται η derived1::f()
    p = &d2;
    p->f(); //καλείται η derived2::f()

    return 0;
}
```

Αποτελέσματα

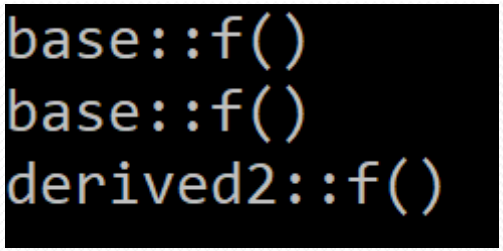
```
base::f()
derived1::f()
derived2::f()
```

Μια εικονική συνάρτηση δεν είναι απαραίτητο να ορίζεται σε όλες τις κλάσεις. Χρησιμοποιείται η εκδοχή της πιο κοντινής κλάσης-προγόνου.

```
class base {
public:
   virtual void f() {
      cout << "base::f()" << endl;
   }
};

class derived1:public base {

};

class derived2:public derived1 {
public:
   void f() {
      cout << "derived2::f()" <<endl;
   }
};
```

```
int main()
{

    base b, *p;
    derived1 d1;
    derived2 d2;

    p = &b;
    p->f(); //καλείται η base::f()
    p = &d1;
    p->f();  //Δεν υπάρχει derived1:f(). Καλείται η base::f()
    p = &d2;
    p->f(); //καλείται η derived2::f()

    return 0;
}
```

Αποτέλεσμα



Άλλο παράδειγμα

```
class base {
public:
   virtual void f() {
      cout << "base::f()" << endl;
   }
};

class derived1:public base {
public:
   void f() {
      cout << "derived1::f()" <<endl;
   }
};

class derived2:public derived1 {

};
```

```
int main()
{

    base b, *p;
    derived1 d1;
    derived2 d2;

    p = &b;
    p->f(); //καλείται η base::f()
    p = &d1;
    p->f();  //καλείται η derived1:f().
    p = &d2;
    p->f(); //Δεν υπάρχει derived2::f(). Καλείται η derived1::f()

    return 0;
}
```

Αποτέλεσμα

```
base::f()
derived1::f()
derived1::f()
```

Καθαρά εικονικές συναρτήσεις

- Υπάρχουν περιπτώσεις που μια εικονική συνάρτηση δεν έχει νόημα να οριστεί στη βασική κλάση.
  - Δεν υπάρχει κάποια γενική ενέργεια που να μπορεί να χρησιμοποιηθεί σε όλες τις περιπτώσεις.
  - Η συνάρτηση δηλώνεται ως εικονική στη βασική κλάση αλλά δεν ορίζεται. Η σύνταξη είναι

    virtual *τύπος όνομα*(*παράμετροι*) = 0;
- Μια τέτοια συνάρτηση ονομάζεται *καθαρά εικονική* συνάρτηση (pure virtual function).
- Μια κλάση που έχει τουλάχιστον μια καθαρά εικονική συνάρτηση-μέλος ονομάζεται *αφηρημένη κλάση* (abstract class).
- Όλες οι κλάσεις-παιδιά μιας αφηρημένης κλάσης πρέπει να ξαναορίζουν τις καθαρά εικονικές συναρτήσεις.
  - Διαφορετικά ο Compiler εμφανίζει μήνυμα σφάλματος.
- Μια αφηρημένη κλάση παριστάνει ουσιαστικά μια διεπαφή προς μια γενική κατηγορία αντικειμένων.
  - Δεν μπορούμε να δημιουργήσουμε στιγμιότυπα από μια αφηρημένη κλάση.
  - Μπορούμε όμως να δηλώνουμε δείκτες ή αναφορές προς αυτήν και να εφαρμόζουμε τον πολυμορφισμό χρόνου εκτέλεσης.

Στατικά μέλη κλάσεων

Ο όρος static στο διαδικαστικό προγραμματισμό

- Στο διαδικαστικό προγραμματισμό (C,C++) μπορούμε γενικά να χαρακτηρίζουμε κάποιες μεταβλητές (ή και συναρτήσεις) ως static.
- Εάν πρόκειται για μεταβλητές (ή συναρτήσεις) γενικής εμβέλειας ο όρος static δηλώνει ότι δεν είναι προσβάσιμες έξω από το αρχείο στο οποίο έχουν δηλωθεί.
- Εάν πρόκειται για τοπικές μεταβλητές κάποιας συνάρτησης, ο όρος static δηλώνει ότι διατηρούν την τιμή τους μεταξύ των διαδοχικών κλήσεων της συνάρτησης.
  - Εάν η δήλωσή τους περιλαμβάνει και αρχική τιμή, η αρχική αυτή τιμή προσδίδεται στις μεταβλητές μόνο κατά την πρώτη κλήση της συνάρτησης.

Στατικές μεταβλητές-μέλη

- Στον αντικειμενοστραφή προγραμματισμό, όταν μια μεταβλητή-μέλος χαρακτηρίζεται ως στατική (static) αυτό σημαίνει ότι η μεταβλητή χαρακτηρίζει συνολικά την κλάση και όχι τα στιγμιότυπά της.
- Υπάρχει ένα και μόνο αντίγραφο της μεταβλητής που το μοιράζονται όλα τα στιγμιότυπα της κλάσης.
  - Παράδειγμα: Μπορεί να χρησιμοποιούμε μια στατική μεταβλητή μέλος για να μετράμε πόσα στιγμιότυπα της κλάσης έχουν δημιουργηθεί.
- Η πρόσδοση αρχικής τιμής (αν χρειάζεται) γίνεται έξω από τον κώδικα της κλάσης χρησιμοποιώντας την σύνταξη

  κλάση::μεταβλητή = τιμή

  Διαφορετικά η μεταβλητή παίρνει αυτόματα αρχική τιμή 0 μόλις δημιουργηθεί το πρώτο στιγμιότυπο της κλάσης.

Στατικές συναρτήσεις μέλη

- Όταν μια συνάρτηση-μέλος χαρακτηρίζεται στατική αφορά συνολικά την κλάση και όχι τα στιγμιότυπά της.
- Μια στατική συνάρτηση-μέλος δεν μπορεί να χρησιμοποιεί μη στατικά μέλη της κλάσης (μεταβλητές ή συναρτήσεις).
  - Μπορεί να χρησιμοποιεί στατικές μεταβλητές-μέλη και να καλεί στατικές συναρτήσεις-μέλη ή συναρτήσεις ορισμένες εκτός της κλάσης.
  - Δεν μπορεί να χρησιμοποιεί τον δείκτη this.
  - Δεν μπορεί να δηλωθεί ως εικονική συνάρτηση