ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ

ΟΡΙΣΜΟΣ

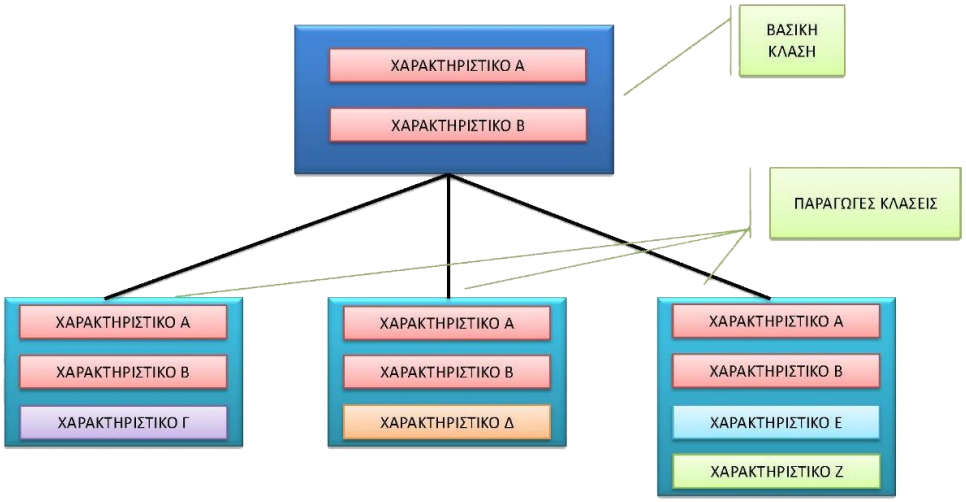
Κληρονομικότητα ονομάζεται η διαδικασία που επιτρέπει σε μία κλάση να **κληρονομεί** τις ιδιότητες (δεδομένα) και τις συμπεριφορές (συναρτήσεις) μιας άλλης κλάσης.

- Η κλάση που κληρονομείται λέγεται **βασική κλάση** (basic class) ή **υπερκλάση** (superclass) ενώ,
- Η κλάση που κληρονομεί λέγεται **παραγόμενη κλάση** (derived class) ή **υποκλάση** (subclass).

ΠΛΕΟΝΕΚΤΗΜΑ ΤΗΣ ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑΣ

Βασικό πλεονέκτημα της κληρονομικότητας είναι ότι χρησιμοποιείται ο ήδη γραμμένος κώδικάς μιας κλάσης, που με κάποιες προσθήκες προσαρμόζεται σε μία νέα κλάση (παραγόμενη κλάση) η οποία κληρονομεί χαρακτηριστικά και μεθόδους, αυτό λέγεται **επαναχρησιμοποίηση κώδικα**.

Έτσι δεν χρειάζεται να ξαναγράψουμε τον κώδικα από την αρχή για τη νέα κλάση. Αυτό έχει σαν αποτέλεσμα να εξοικονομούμε χρόνο και χρήμα, αλλά και να καθιστούμε το πρόγραμμα πιο αξιόπιστο και εύκολα αναγνώσιμο.



ΣΥΝΤΑΞΗ ΒΑΣΙΚΗΣ ΚΑΙ ΠΑΡΑΓΟΜΕΝΗΣ ΚΛΑΣΗΣ

Ο ορισμός της παραγόμενης κλάσης στην C++ γίνεται ως εξής:

```
class παράγωγη_Κλάση : public βασική_Κλάση
```

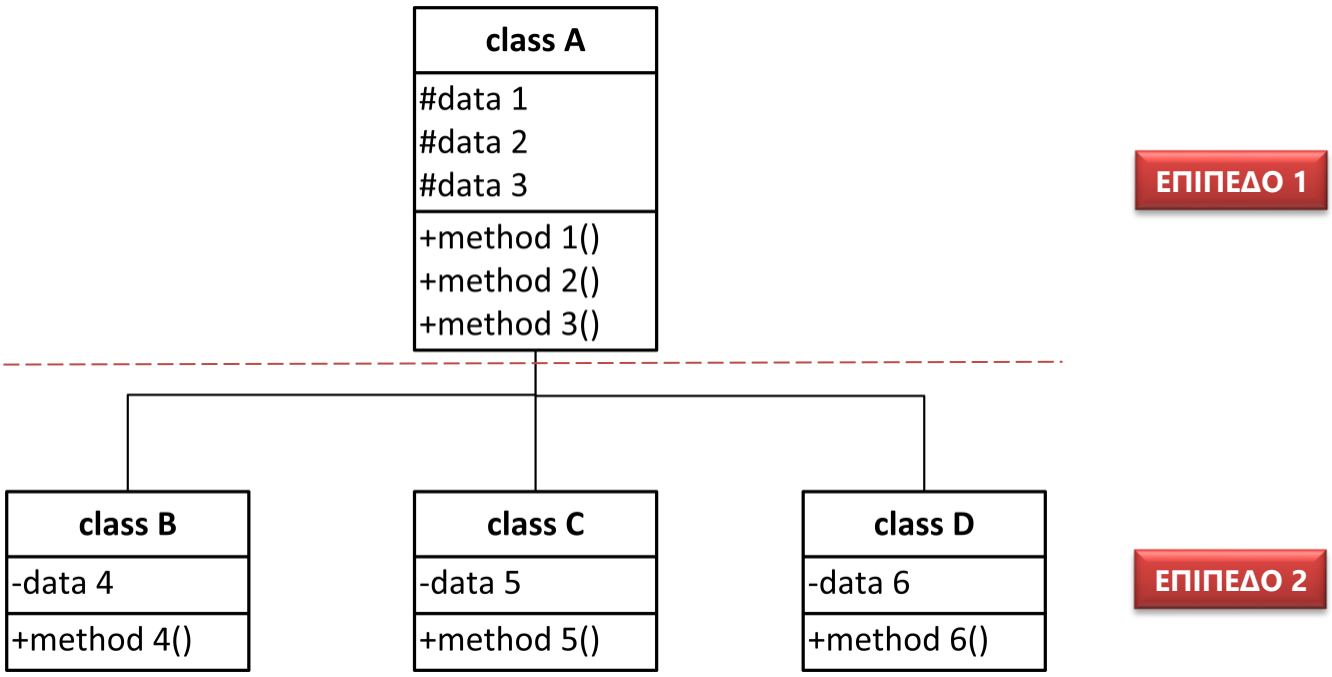
Η παράγωγη κλάση κληρονομεί και τα χαρακτηριστικά αλλά και τις μεθόδους της βασικής κλάσης αφού παράγεται δημόσια (public) από την βασική.

ΠΡΟΣΤΑΤΕΥΟΜΕΝΑ ΔΕΔΟΜΕΝΑ

Ο τρόπος πρόσβασης στα δεδομένα μέλη από τις μεθόδους της κλάσης στην κληρονομικότητα αλλάζει λίγο. Τα δεδομένα στην βασική κλάση πρέπει να δηλωθούν ως **προστατευμένα** - δεδομένα μέλη με την χρήση της δεσμευμένης από τη γλώσσα λέξης **protected** (στην UML με το σύμβολο #). Αυτό γίνεται για να μπορούν να έχουν πρόσβαση σε αυτά (ή αλλιώς να τα προσπελάσουν), οι συναρτήσεις-μέλη της ίδιας της κλάσης αλλά και οι συναρτήσεις - μέλη των παραγόμενων κλάσεων. Σε αντίθεση με την δήλωση private όπου μπορεί να έχει πρόσβαση στα δεδομένα, μόνο η κλάση στην οποία ανήκουν.

Tip 1. *Δεν μπορούν να έχουν πρόσβαση στα δεδομένα, συναρτήσεις που βρίσκονται έξω από αυτές τις κλάσεις δηλ. στην main().*

ΠΑΡΑΔΕΙΓΜΑ ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑΣ ΣΕ UML



Στο παραπάνω διάγραμμα κληρονομικότητας δύο επιπέδων φαίνονται οι παράγωγες κλάσεις class B, class C, class D οι οποίες κληρονομούν από την βασική κλάση class A τα δεδομένα μέλη data 1, data 2, data 3 καθώς και τις μεθόδους της βασικής κλάσης method 1(), method 2(), method 3(). Εδώ λοιπόν βλέπουμε ότι:

- Τα δεδομένα της βασικής κλάσης δηλώνονται protected (#) δηλαδή προστατευμένα ενώ οι μέθοδοι public (+) δηλαδή δημόσια ορατές.
- Οι παραγόμενες κλάσεις διαθέτουν και δικά τους δεδομένα που δηλώνονται private (-) και μεθόδους που δηλώνονται public (+).
- Τα αντικείμενα που θα προκύψουν από τις παραγόμενες κλάσεις θα έχουν πρόσβαση και στα δεδομένα αλλά και στις μεθόδους της βασικής κλάσης.
- Για παράδειγμα ένα αντικείμενο της παράγωγης κλάσης class D έχει τέσσερα δεδομένα δηλαδή τα: data1 , data 2, data 3 (της βασικής κλάσης) και data 4 (της παράγωγης κλάσης) στα οποία μπορούν να έχουν πρόσβαση τέσσερις μέθοδοι δηλαδή οι: method 1, method 2, method 3 (της βασικής κλάσης) και method 4 (της παράγωγης κλάσης).

ΔΗΛΩΣΗ ΠΡΟΣΒΑΣΙΜΟΤΗΤΑΣ ΣΕ ΚΛΑΣΗ

```
class basic_class
{
    protected:
            int mesure;
    public:
            basic_class()
            {
                  mesure = 0;
            }
};
```

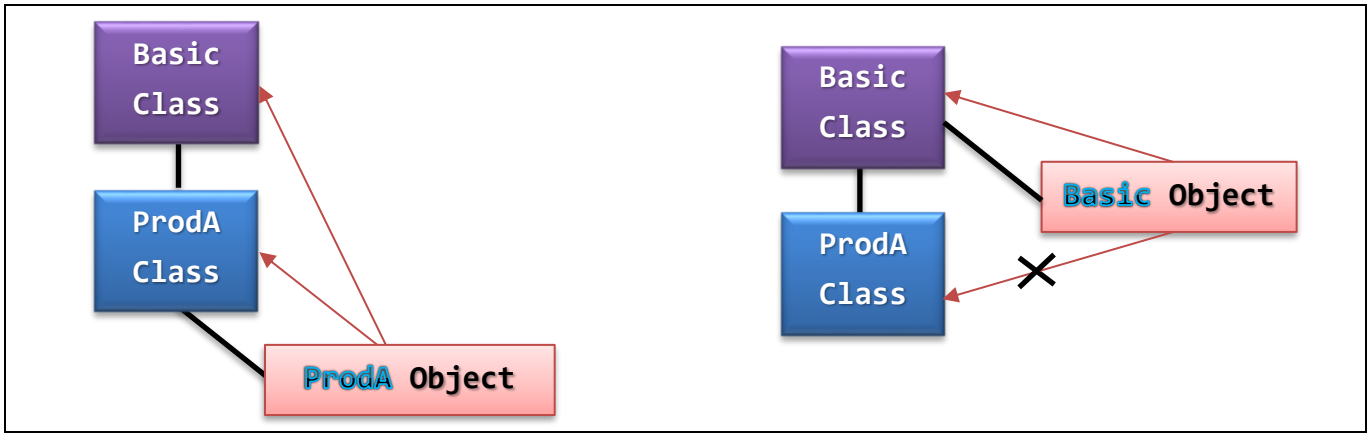
Χώρος προστατευμένης πρόσβασης ο οποίος είναι ορατός στην βασική κλάση αλλά και στις παράγωγες κλάσεις

Tip 2. *Στην βασική κλάση τα δεδομένα μέλη ορίζονται στον χώρο προστατευμένης πρόσβασης έτσι ώστε να είναι ορατά και από τα αντικείμενα των παράγωγων κλάσεων.*

ΑΝΤΙΣΤΡΟΦΑ ΔΟΥΛΕΥΕΙ;

ΠΡΟΣΟΧΗ!!! Η κληρονομικότητα **δεν λειτουργεί αντίστροφα**.

Δηλαδή εάν φτιάξουμε στην main() ένα αντικείμενο της βασικής κλάσης, αυτό θα έχει πρόσβαση μόνο στις μεθόδους της βασικής κλάσης και όχι στις μεθόδους που ανήκουν στην παραγόμενη κλάση.



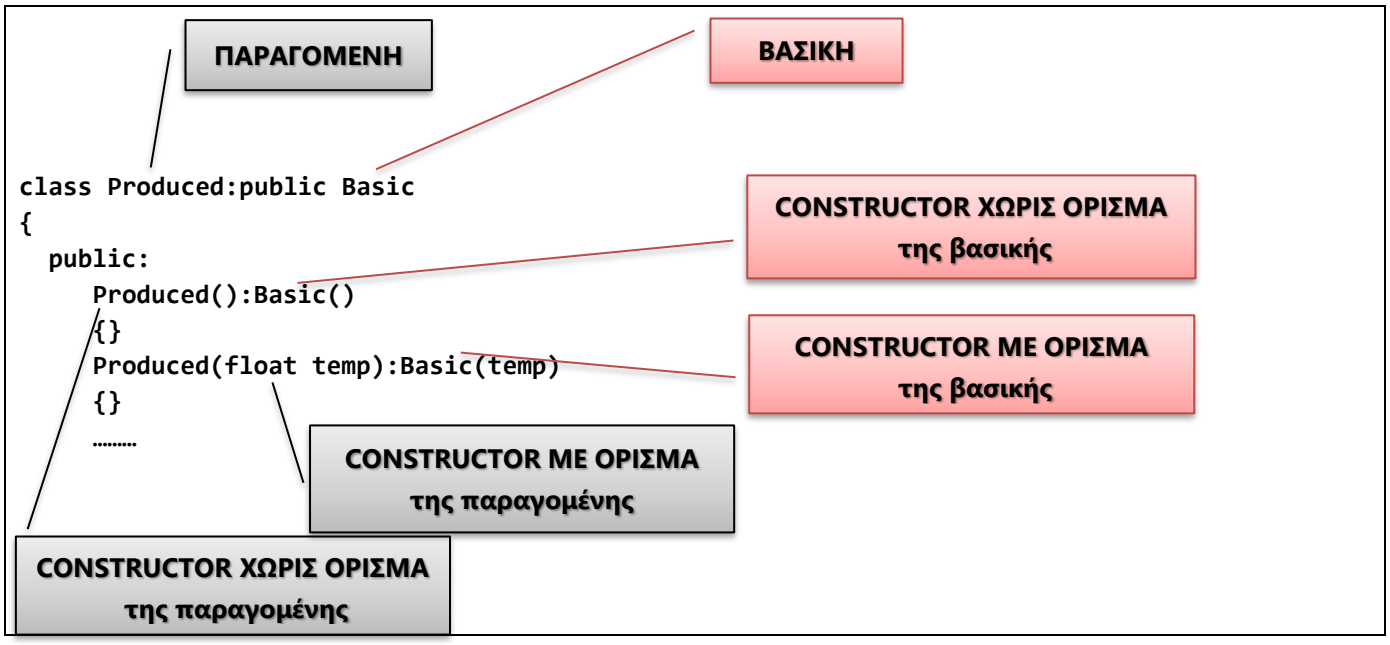
CONSTRUCTOR ΣΤΗΝ ΠΑΡΑΓΟΜΕΝΗ ΚΛΑΣΗ

Αν και μπορούμε να χρησιμοποιήσουμε τον constructor δίχως όρισμα της βασικής κλάσης για να αρχικοποιήσουμε ένα αντικείμενο της παραγόμενης κλάσης, το ίδιο δεν μπορεί να συμβεί σε constructor με όρισμα της βασικής κλάσης. Σε αυτή την περίπτωση φτιάχνουμε constructors με όρισμα και χωρίς όρισμα και στην παραγόμενη κλάση.

Έτσι για αρχικοποίηση από την παραγόμενη κλάση καλούμε τον constructor της βασικής.

Tip 3*. χρησιμοποιούμε υπερφόρτωση κατασκευαστών και στην βασική αλλά και στην παραγόμενη κλάση.*

ΠΑΡΑΔΕΙΓΜΑ CONSTRUCTOR ΒΑΣΙΚΗΣ ΚΑΙ ΠΑΡΑΓΟΜΕΝΗΣ ΚΛΑΣΗΣ



Όταν στην main() φτιάξουμε ένα αντικείμενο της παραγόμενης χωρίς όρισμα δηλ. το **Produced Pr1** τότε εκτελείται η εντολή:

```
Produced():Basic()
```

Και ο constructor Produced() καλεί τον constructor Basic() για την αρχικοποίηση του Pr1.

Αντίστοιχα εάν φτιάξουμε ένα αντικείμενο με όρισμα δηλ. το **Produced Pr2(20.5),** τότε εκτελείται η εντολή:

```
Produced(float temp):Basic(temp)
```

όπου καλείται ο constructor Produced(float temp) με όρισμα και αυτός καλεί τον constructor Basic(temp) με όρισμα και του μεταβιβάζει την παράμετρο temp για να αρχικοποιηθεί το αντικείμενο Pr2 με τιμή 20,5.

ΠΑΡΑΔΕΙΓΜΑΤΑ ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑΣ

Παράδειγμα 1. Δήλωση κληρονομικότητας κλάσεων δύο επιπέδων.

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

//Δήλωση Βασικής Κλάσης
class Accounts{

	protected: //Χώρος προστατευμένης πρόσβασης
		float balance;
	public:
		//Δήλωση constructor χωρίς ορίσματα
		Accounts(){
			balance = 0;
		}

		//Δήλωση constructor με όρισμα
		Accounts(float aBalance){
			balance = aBalance;
		}

		//Δήλωση destructor
		~Accounts(){}

		//Ορισμός μεθόδου ανάληψης
		void withdrawal(float amount){
		   if (amount <= balance)
		       balance -= amount;
		   else{
		       cout<<"ATTENTION!!!!!!"<<endl;
		       cout<<"THE AMOUNT IS GREATER THAN THE BALANCE OF THE ACCOUNT"<<endl;
		   }
		}

		//Ορισμός μεθόδου κατάθεσης
		void deposit(float amount){
			balance += amount;
		}

		//Ορισμός μεθόδου επιστροφής υπολοίπου
		float returnBalance(){
			return balance;
		}

}; //Τέλος Βασικής Κλάσης
```

```
//Δήλωση Παραγόμενης Κλάσης
class Interest:public Accounts{

      public:
            //μέθοδος παραγόμενης κλάσης
            void interest_payment(){
                balance += balance * 0.1; //Απόδοση τόκου 10%
            }

}; //Τέλος Παραγόμενης Κλάσης


int main(){
       Interest AI1; //Αντικείμενο παραγόμενης κλάσης

       //Έλεγχος υπολοίπου
       cout << "Balance is: " << AI1.returnBalance() << " Euros" << endl;

       //Κατάθεση από το αντικείμενο της παραγόμενης κλάσης
       cout << "\nDeposit 120 Euros" << endl;
       AI1.deposit(120.0);

       //Έλεγχος νέου υπόλοιπου
       cout << "New Balance is: " << AI1.returnBalance() << " Euros" << endl;

        //Απόδοση τόκου στο αντικείμενο της παραγόμενης κλάσης
       cout << "\nInterest Payment 10%" << endl;
       AI1.interest_payment();

       //Έλεγχος νέου υπόλοιπου
       cout << "New Balance is: " << AI1.returnBalance() << " Euros" << endl;
} //Τέλος του προγράμματος
```

Σύντομη εξήγηση του προγράμματος.

Στο πιο πάνω παράδειγμα δημιουργήσαμε μια βασική κλάση **Accounts** και μία δημόσια παραγόμενη κλάση **Interest.** Στην **main()** δημιουργούμε ένα αντικείμενο **AI1** της παράγωγης κλάσης Interest. Το αντικείμενο αρχικοποιείται στην τιμή 0 χρησιμοποιώντας τον constructor της βασικής κλάσης.

Επίσης το αντικείμενο μπορεί να χρησιμοποιεί τις μεθόδους **deposit()** και **returnBalance()** της κλάσης Accounts. Το δεδομένο-μέλος έχει δηλωθεί ως **protected** έτσι ώστε να είναι προσπελάσιμο και από την βασική κλάση στην οποία ανήκει αλλά και στην παράγωγη (εάν ήταν private θα είχε πρόσβαση σε αυτό μόνο η κλάση στην οποία ανήκει). Η κληρονομικότητα δεν λειτουργεί αντίστροφα, δηλαδή εάν φτιάξουμε στην **main()** ένα αντικείμενο της βασικής κλάσης π.χ. **Accounts AC1** αυτό θα έχει πρόσβαση μόνο στις μεθόδους της κλάσης στην οποία ανήκει που είναι η βασική αλλά όχι και στην μέθοδο **interest_payment()** που ανήκει στην παραγόμενη κλάση.

Παράδειγμα 2. Δήλωση και χρήση των συναρτήσεων εγκατάστασης (constructor) σε κληρονομικότητα κλάσεων δύο επιπέδων.

```
#include <iostream.h>
#include <string.h>

class base_class{
  protected:
    float data1;
    int data2;
  public:
    base_class(){
       data1 = 0;
       data2 = 0;
    }
    base_class(float x, int y){
       data1 = x;
       data2 = y;
    }
……
……
class prod_class: public base_class{
   private:
      char data_p[10];
   public:
     prod_class(){
        strcpy(data_p = "No Data");
     }
     prod_class(float  a, int b, char c[]):base_class(a, b){
       strcpy(data_p,c);
     }
……
……

int main(){
     base_class bc1, bc2(3.2, 9);
     prod_class pc1, pc2(4.3, 4, "hello");

……
}
```

Σύντομη εξήγηση του προγράμματος.

Πιο πάνω βλέπουμε την δήλωση και χρήση των συναρτήσεων εγκατάστασης (constructor) και στην παράγωγη κλάση **prod_class** αλλά και στην βασική **base_class.**

Η διαδικασία αρχικοποίησης των δεδομένων από τους constructor είναι η εξής:

- Το αντικείμενο **bc2** της βασικής κλάσης αρχικοποιεί τα **data1** και **data2** στέλνοντας τις τιμές τους κατευθείαν στον constructor base_class() της βασικής.

Έτσι το αντικείμενο bc2 θα έχει κατά την κατασκευή του τις τιμές data1 -> **3.2** και data2 -> **9 .**

- Στην παραγόμενη κλάση που έχει και αυτή ένα δεδομένο μέλος για αρχικοποίηση το **data_p**, το αντικείμενο **pc2** στέλνει τρεις τιμές στον constructor prod_class() της παραγόμενης κλάσης,
- οι δύο πρώτες ανήκουν στα δεδομένα της βασικής κλάσης και η τελευταία στο δεδομένο της παραγόμενης κλάσης.
- Ο constructor prod_class() της παραγόμενης κλάσης στέλνει τις δύο πρώτες τιμές για αρχικοποίηση στον constructor base_class() της βασικής και κρατά το τελευταίο για να το αρχικοποιήσει αυτός.

Έτσι το αντικείμενο pc2 θα έχει κατά την κατασκευή του τις τιμές data1 -> **4.3**, data2 -> **4** και data_p -> **"hello".**

Tip 4*. τα αντικείμενα* ***bc1*** *και* ***pc1*** *θα αρχικοποιηθούν με τις τιμές που τους δίνουν οι constructors χωρίς ορίσματα με την ίδια λογική .*

Πιο κάτω ακολουθεί το πρόγραμμα του παραδείγματος 2 ολοκληρωμένο.

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

//Δήλωση Βασικής Κλάσης
class base_class{

    protected: //Χώρος προστατευμένης πρόσβασης
        float data1;
        int data2;
    public:

        //Δήλωση constructor χωρίς ορίσματα
        base_class(){
            data1 = 0;
            data2 = 0;
        }

        //Δήλωση constructor με ορίσματα
        base_class(float x, int y){
            data1 = x;
            data2 = y;
        }

        //Δήλωση Destructor
        ~base_class(){
        }

        //Ορισμός μεθόδου εκτύπωσης τιμών
        void printBaseClassInfos(){
            cout << "Data1 = " << data1 << "  Data2 = " << data2 << endl;
        }
}; //Τέλος Βασικής Κλάσης


//Δήλωση Παραγόμενης Κλάσης
class prod_class: public base_class{

   private: //Χώρος ιδιωτικής πρόσβασης
       char data_p[10];
   public:
        //Δήλωση constructor χωρίς όρισμα
        prod_class(){
            strcpy(data_p, "NoName");
        }

        //Δήλωση constructor με όρισμα
        prod_class(float  a, int b, char c[]):base_class(a, b){
            strcpy(data_p,c);
        }

        //Δήλωση Destructor
        ~prod_class(){
        }

        //Ορισμός μεθόδου εκτύπωσης τιμών
        void printProdClassInfos(){
            cout << "Data1 = " << data1 << "  Data2 = " << data2 << "  Data3 = "<< data_p <<
endl;
        }
}; //Τέλος Παραγόμενης Κλάσης
```

```
int main(){

    //Αντικείμενα βασικής κλάσης
    base_class bc1, bc2(3.2, 9);

    //Αντικείμενα παραγόμενης κλάσης
    prod_class pc1, pc2(4.3, 4, "hello");

    //Εκτύπωση των τιμών των αντικειμένων της βασικής κλάσης
    cout << "\n====First Object from Superclass====" << endl;
    bc1.printBaseClassInfos();

    cout << "\n====Second Object from Superclass====" << endl;
    bc2.printBaseClassInfos();


    //Εκτύπωση των τιμών των αντικειμένων της παραγόμενης κλάσης
    cout << "\n====First Object from Subclass====" << endl;
    pc1.printProdClassInfos();

    cout << "\n====Second Object from Subclass====" << endl;
    pc2.printProdClassInfos();

    return 0;
} //Τέλος του προγράμματος
```