ΤΕΙ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΑΝΟΙΚΤΑ ΑΚΑΔΗΜΑΙΚΑ ΜΑΘΗΜΑΤΑ

ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ(Θ)

Ενότητα 2: ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

ΔΙΔΑΣΚΩΝ: ΠΑΡΙΣ ΜΑΣΤΟΡΟΚΩΣΤΑΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΤΕ

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.

- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο ΤΕΙ Κεντρικής Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ενότητα 2

ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

ΔΙΔΑΣΚΩΝ: ΠΑΡΙΣ ΜΑΣΤΟΡΟΚΩΣΤΑΣ

Περιεχόμενα ενότητας

1. Κλάσεις και αντικείμενα
2. Ορισμός αντικειμένων
3. Ιδιωτικά και δημόσια μέλη
4. Κλήση συναρτήσεων-μελών
5. Απόδοση τιμών μέσω παραμέτρων
6. Συναρτήσεις εγκατάστασης/δόμησης (constructors)
7. Συναρτήσεις αποσύνδεσης/αποδόμησης (destructors)
8. Συναρτήσεις δόμησης με υπερφόρτωση (constructor overloading)
9. Συναρτήσεις-μέλη οριζόμενες έξω από την κλάση
10. Συναρτήσεις δόμησης – εναλλακτικός ορισμός
11. Αντικείμενα ως ορίσματα συναρτήσεων
12. Επιστροφή αντικειμένων από συναρτήσεις
13. Ένταξη κλάσης στους τύπους δεδομένων

Σκοποί ενότητας

Κλάσεις και αντικείμενα

```
#include <iostream.h>
class Person
{
   private:
            char name[30];
            int age;
   public:
            void readData()
            {
               cout << “Enter name:”;
               cin >> name;
               cout << “Enter age:”;
               cin >> age;
            }
            void printData()
            {
               cout << “The name of the person is “ << name << endl;
               cout << “The age of the person is “ << age << endl;
            }
};          // τέλος κλάσης
```

Κλάσεις και αντικείμενα

```
void main()
{
   Person  p1, p2;          // δήλωση δύο αντικειμένων

   p1.readData();     // κλήση συνάρτησης μέλους για ορισμό δεδομένων
   p1.printData();    // κλήση συνάρτησης μέλους για εμφάνιση δεδομένων
   p2.readData();
   p2.printData();
}
```

Ορισμός αντικειμένων

Η πρόταση

Person p1, p2;

ορίζει δύο **αντικείμενα** (objects) της κλάσης **Person**.

Θα πρέπει να τονισθεί ότι, όπως σε μία δομή, όταν την ορίζουμε δε δημιουργούμε καμία μεταβλητή δομής, έτσι και με την κλάση, όταν την ορίζουμε δε δημιουργούμε αντικείμενα, απλά περιγράφουμε πώς θα είναι αυτά όταν δημιουργηθούν. Τα αντικείμενα δημιουργούνται όταν ορίζονται, σύμφωνα με την παραπάνω πρόταση.

Ιδιωτικά και δημόσια μέλη

Ένα βασικό χαρακτηριστικό του αντικειμενοστραφούς προγραμματισμού είναι η απόκρυψη δεδομένων (*data hiding*).

- **Η ένδειξη private (*ιδιωτικά*) σημαίνει ότι τα δεδομένα είναι κρυμμένα μέσα σε μία κλάση, έτσι ώστε να μην είναι προσπελάσιμα από συναρτήσεις που είναι έξω από την κλάση.**
- **Αντιθέτως, τα δημόσια (*public*) δεδομένα, προσπελαύνονται και έξω από τις κλάσεις.**

Συνήθως τα δεδομένα (ή τουλάχιστον τα περισσότερα εξ αυτών) μέσα σε μία κλάση είναι ιδιωτικά και οι συναρτήσεις είναι δημόσιες.

Κλήση συναρτήσεων-μελών

Για να καλέσουμε μία συνάρτηση-μέλος μίας κλάσης τη συνδέουμε μέσω του τελεστή τελεία με κάποιο αντικείμενο:

Αντικείμενο.συνάρτηση-μέλος

Η σύνταξη είναι παρόμοια με τον τρόπο που αναφερόμαστε στα μέλη μίας δομής, με τις παρενθέσεις να υποδηλώνουν όμως ότι αναφερόμαστε σε συνάρτηση-μέλος και όχι σε στοιχείο δεδομένων.

Η πρώτη κλήση στη readData() γίνεται ως εξής:

p1.readData();

όπου καλείται η συνάρτηση-μέλος readData() του αντικειμένου p1.

Ακολούθως, εκτελείται ο κώδικας της συνάρτησης, όπου δίνονται από τον χρήστη τιμές για το όνομα και την ηλικία του ατόμου. Οι τιμές αποδίδονται στα δεδομένα του αντικειμένου name και age, αντίστοιχα.

Η δεύτερη κλήση

p2.readData();

καλεί p2.

Απόδοση τιμών μέσω παραμέτρων

```
class Person
{
  private:
     char name[30];
     int age;
  public:
     void setData(char name1[], int age1)
     {
         strcpy(name, name1);
         age = age1;
     }
     void printData()
     {
        cout << “The name of the person is “ << name << endl;
         cout << “The age of the person is “ << age << endl;
   }
};          // τέλος κλάσης
```

Εφόσον τα ιδιωτικά μέλη δηλώνονται πριν τα δημόσια, η λέξη private περιττεύει.

Απόδοση τιμών μέσω παραμέτρων

```
void main()
{
   Person  p;
   p.setData("PAPADOPOULOS", 25);
   p.printData();
}
```

Στο πρόγραμμα αυτό ορίζεται ένα αντικείμενο p. Ακολούθως, με την πρόταση

p.setData("PAPADOPOULOS", 25);

καλείται η συνάρτηση-μέλος setData(), στην οποία μεταβιβάζονται οι τιμές "PAPADOPOULOS " και 25 στις παραμέτρους name1 και age1, αντίστοιχα.

Τέλος, όταν εκτελείται ο κώδικας της συνάρτησης, οι τιμές αυτών των παραμέτρων αποδίδονται στα δεδομένα του αντικειμένου.

Συναρτήσεις εγκατάστασης/δόμησης (constructors)

•Μερικές φορές είναι βολικό για ένα αντικείμενο να λαμβάνει άμεσα αρχικές τιμές, την πρώτη φορά που δημιουργείται, χωρίς να απαιτείται να καλέσουμε μία ξεχωριστή συνάρτηση-μέλος.

•Η αυτόματη απόδοση αρχικών τιμών πραγματοποιείται με χρήση μίας ειδικής συνάρτησης-μέλους που ονομάζεται *συνάρτηση εγκατάστασης ή δόμησης ή δομητής (constructor)*.

•Η συνάρτηση δόμησης είναι μία συνάρτηση-μέλος της κλάσης, η οποία υποχρεωτικά είναι είναι συνονόματη με thn κλάση. Καλείται αυτόματα όταν δηλώνεται ένα αντικείμενο της κλάσης. Οι δομητές χρησιμοποιούνται για την αρχικοποίηση αντικειμένων. Ένας δομητής δεν επιστρέφει τιμή ούτε δηλώνεται ως void.

Συναρτήσεις εγκατάστασης/δόμησης (constructors)

```
class Account
{
   private:
    float  balance;
   public:
    Account()
    {
       balance = 0;
    }
    void  withdraw(float  money)
    {
       if (money <= balance)
          balance = balance - money;
       else
          cout << "Το ποσό ανάληψης υπερβαίνει το τρέχον!" << endl;
    } // συνέχεια της κλάσης στην επόμενη διαφάνεια
```

constructor

Συναρτήσεις εγκατάστασης/δόμησης (constructors)

```
    void deposit(float  money)
    {
        balance += money;
    }
    float getBalance()
    {
        return  balance;
    }
};      // τέλος της κλάσης
main()
{
    Account  ac;
    ac.deposit(100.0);
    cout << "Τρέχον ποσό λογαριασμού:" << ac.getBalance() << endl;
    ac.withdraw(70.0) ;
    cout << "Τρέχον ποσό λογαριασμού:" << ac.getBalance() << endl;
 }
```

Συναρτήσεις εγκατάστασης/δόμησης (constructors)

- Στο πρόγραμμα ορίζεται μία κλάση Account, με ένα μέλος δεδομένων balance, που αναφέρεται στο τρέχον ποσό ενός λογαριασμού.

- Ορίζονται επίσης δύο συναρτήσεις-μέλη για την ανάληψη και την κατάθεση χρημάτων στο λογαριασμό και μία συνάρτηση-μέλος που επιστρέφει, κατά την κλήση της, το τρέχον ποσό λογαριασμού.

- Το τρέχον ποσό λογαριασμού αρχικοποιείται στο 0. Αυτό επιτυγχάνεται κατά τη δημιουργία ενός αντικειμένου, με τη συνάρτηση εγκατάστασης Account(), η οποία εκτελείται αυτόματα.

Συναρτήσεις αποσύνδεσης/αποδόμησης (destructors)

```
class Account
{
   private:
    float  balance;
   public:
    Account()
    {
       balance = 0;
    }
    ~Account()          // συνάρτηση αποσύνδεσης
    {
       cout << "The account has been deleted" << endl;
    }
    .
    .
};
```

Συναρτήσεις αποσύνδεσης/αποδόμησης (destructors)

- Μία συνάρτηση δόμησης καλείται αυτόματα όταν δημιουργείται ένα αντικείμενο.
- Μπορούμε αντίστοιχα να ορίσουμε μία συνάρτηση αποσύνδεσης ή αποδόμησης ή αποδομητή (*destructor*), η οποία θα καλείται αυτόματα όταν ένα αντικείμενο καταστρέφεται.
- Η συνάρτηση αποδόμησης έχει το ίδιο όνομα με την κλάση, έχοντας μπροστά μία περισπωμένη ~.
- Η πιο συνηθισμένη χρήση των συναρτήσεων αποδόμησης είναι η αποδέσμευση της μνήμης που είχε δεσμευτεί για ένα αντικείμενο από τη συνάρτηση δόμησης.

Συναρτήσεις δόμησης με υπερφόρτωση (*constructor overloading*)

```
class Account
{
   private:
            float  balance;
   public:
            Account()  // συνάρτηση δόμησης χωρίς ορίσματα
            {
               balance = 0;
            }
            Account(float  balance1)     // συνάρτηση δόμησης με όρισμα
            {
               balance = balance1;
            }
            void  withdraw(float  money)
            {
               if (money <= balance)
                  balance = balance - money;
               else
                  cout << "Το ποσό ανάληψης υπερβαίνει το τρέχον!" << endl;
            }
```

Συναρτήσεις δόμησης με υπερφόρτωση (*constructor overloading*)

```
    void deposit(float  money)
    {
       balance += money;
    }
    float getBalance()
    {
       return  balance;
    }
};
main()
{
   Account  ac1, ac2(50.0), ac3(100.0);
   cout << "Τρέχον ποσό λογαριασμού ac1:" << ac1.getBalance() << endl;
   cout << "Τρέχον ποσό λογαριασμού ac2:" << ac2.getBalance() << endl;
   cout << "Τρέχον ποσό λογαριασμού ac3:" << ac3.getBalance() << endl;
 }
```

Συναρτήσεις δόμησης με υπερφόρτωση (*constructor overloading*)

- Δύο συναρτήσεις δόμησης, μία με ορίσματα και μία χωρίς όρισμα.
- Κατά τη δήλωση

```
Account ac1, ac2(50.0), ac3(100.0);
```

δημιουργούνται τρία αντικείμενα.

- Στο πρώτο γίνεται χρήση της συνάρτησης δόμησης χωρίς όρισμα και που ουσιαστικά δεν κάνει τίποτε.
- Στα δύο επόμενα αντικείμενα γίνεται χρήση της συνάρτησης δόμησης με το όρισμα balance1, στο οποίο μεταβιβάζονται οι τιμές 50.0 και 100.0, αντίστοιχα, και αποδίδονται στα στοιχεία δεδομένων των δύο αντικειμένων.
- Οι δύο συναρτήσεις έχουν το ίδιο όνομα - το όνομα της κλάσης - και θεωρούμε ότι η συνάρτηση δόμησης έχει υποστεί *υπερφόρτωση* (*overloading*).
- Το ποια συνάρτηση δόμησης εκτελείται όταν δημιουργείται ένα αντικείμενο, εξαρτάται από τον αριθμό των ορισμάτων που χρησιμοποιούνται στον ορισμό του αντικειμένου.

Συναρτήσεις-μέλη οριζόμενες έξω από την κλάση

- Μπορούμε να διατηρήσουμε μέσα στην κλάση τις δηλώσεις όλων των συναρτήσεων μελών, αλλά να τις ορίσουμε έξω από την κλάση.

- Αυτό θα ήταν ίσως προτιμότερο σε μία εφαρμογή που ο αριθμός των συναρτήσεων-μελών είναι μεγάλος, γιατί θα μας βοηθούσε να ελέγχουμε καλύτερα την κλάση. Σε μια τέτοια όμως περίπτωση, θα πρέπει στον ορισμό της συνάρτησης να αναφέρουμε και την κλάση στην οποία ανήκει.

Συναρτήσεις-μέλη οριζόμενες έξω από την κλάση

```
class Account
{
   private:
        float  balance;
   public:
        Account();
        Account(float  balance1);
        void  withdraw(float  money);
        void deposit(float  money);
        float getBalance();
        Account  addBalance(Account  ac);
};
```

2 συναρτήσεις δόμησης

Συναρτήσεις-μέλη οριζόμενες έξω από την κλάση

```
Account::Account()
{
   balance = 0;
}
Account::Account(float  balance1)  // συνάρτηση δόμησης με όρισμα
{
   balance = balance1;
}
void  Account::withdraw(float  money)
{
   if (money <= balance)
      balance = balance - money;
   else
      cout << “Το ποσό ανάληψης υπερβαίνει το τρέχον!” << endl;
}
```

Συναρτήσεις-μέλη οριζόμενες έξω από την κλάση

```
void Account::deposit(float  money)
{
   balance += money;
}
float Account::getBalance()
{
   return  balance;
}
Account  Account::addBalance(Account  ac)
{
   Account  temp;
   temp.balance = balance + ac.balance;
   return temp;
}
```

Συναρτήσεις-μέλη οριζόμενες έξω από την κλάση

```
main()
{
   Account  ac1(100.0), ac2(70.0), ac3;
   ac3 = ac1.addBalance(ac2);
   cout << “Τρέχον ποσό λογαριασμού ac1:” << ac1.getBalance() << endl;
   cout << “Τρέχον ποσό λογαριασμού ac2:” << ac2.getBalance() << endl;
   cout << “Συνολικό ποσό λογαριασμών:” << ac3.getBalance() << endl;
}
```

Σε αυτόν τον ορισμό μίας συνάρτησης παρεμβάλλεται το όνομα της κλάσης και το σύμβολο ::, που ονομάζεται τελεστής διάκρισης ή αναγωγής εμβέλειας (*scope resolution operator*), ο οποίος παρέχει έναν τρόπο για να καθορίζουμε σε ποια κλάση ανήκει μία συνάρτηση.

Συναρτήσεις δόμησης - εναλλακτικός ορισμός

Εναλλακτικός τρόπος ορισμού της συνάρτησης δόμησης:

```
Account :: Account(float balance1) : balance(balance1)
{  /* εσκεμμένα κενό σώμα της συνάρτησης δόμησης */ }
```

Δηλαδή οι τιμές με τις οποίες θα αρχικοποιηθούν οι μεταβλητές-μέλη μπαίνουν σε παρενθέσεις μετά τα ονόματα των μεταβλητών μελών, ενώ αριστερά αυτών τίθεται ο προσδιοριστής :
Εάν στην κλάση υπάρχουν δύο ή περισσότερες μεταβλητές-μέλη (π.χ. float balance, float interest), οι οποίες θα λάβουν τις αρχικές τιμές balance1 και interest1, αντίστοιχα, τότε ο εναλλακτικός τρόπος δόμησης θα έδινε:

```
Account :: Account(float balance1) : balance(balance1), (interest1)
{  /* εσκεμμένα κενό σώμα της συνάρτησης δόμησης */ }
```

Συναρτήσεις δόμησης - εναλλακτικός ορισμός

Το σώμα της συνάρτησης δόμησης μπορεί να περιέχει κώδικα π.χ. έλεγχο κατά πόσον το επιτόκιο είναι θετικό:

```
Account :: Account(float balance1) : balance(balance1),
(interest1)
{
        if (interest < 0 )
        {
                cout << “Error!! Negative interest value\n\n”;
                exit(1);
        }
}
```

Συνολικό παράδειγμα κλάσης με έμφαση στους δομητές – αποδομητές:

Walter Savitch, Absolute C++, 5th ed.

pp. 286-291

Τα δεδομένα της κλάσης BankAccount είναι το υπόλοιπο του λογαριασμού και το επιτόκιο. Το υπόλοιπο απεικονίζεται με δύο ακέραιες μεταβλητές: το ακέραιο μέρος ($) και το κλασματικό μέρος (¢). Η συγκεκριμένη απεικόνιση αποσκοπεί στο να δείξει ότι η εσωτερική αναπαράσταση της έννοιας του κάθε δεδομένου δεν είναι αναγκαίο να ταυτίζεται με μία μόνο μεταβλητή. Επιπλέον, εάν το υπόλοιπο περιγραφόταν από έναν αριθμό κινητής υποδιαστολής θα αποτελούσε μία προσέγγιση της πραγματικής τιμής και μάλιστα με περισσότερα των 2 δεκαδικών ψηφίων, γεγονός που θα ερχόταν σε αντίθεση με την έννοια του cent.

Η κλάση έχει τέσσερις ιδιωτικές συναρτήσεις-μέλη: dollarsPart(), centsPart(), round() και fraction(). Έχουν καταστεί ιδιωτικές γιατί ο σκοπός της χρήσης τους είναι ο ορισμός άλλων συναρτήσεων-μελών.

W. Savitch, Absolute C++, 5th Ed.

Στο συγκεκριμένο παράδειγμα οι ιδιωτικές μεταβλητές-μέλη και συναρτήσεις-μέλη δηλώνονται στο τέλος της κλάσης.

```
1  #include <iostream>
2  #include <cmath>
3  #include <cstdlib>
4  using namespace std;

5  //Data consists of two items: an amount of money for the account balance
6  //and a percentage for the interest rate.
7  class BankAccount
8  {
9  public:
10     BankAccount(double balance, double rate);
11     //Initializes balance and rate according to arguments.

12     BankAccount(int dollars, int cents, double rate);
13     //Initializes the account balance to $dollars.cents. For a
14     //negative balance both dollars and cents must be negative.
       //Initializes the interest rate to rate percent.
```

W. Savitch, Absolute C++, 5th Ed.

```
15      BankAccount(int dollars, double rate);
16      //Initializes the account balance to $dollars.00 and
17      //initializes the interest rate to rate percent.

18      BankAccount( );
19      //Initializes the account balance to $0.00 and the interest rate
        //to 0.0%.
20      void update( );
21      //Postcondition: One year of simple interest has been added to the
        //account.
22      void input( );
23      void output( );
24      double getBalance( );
25      int getDollars( );
26      int getCents( );
27      double getRate( );//Returns interest rate as a percentage.

28      void setBalance(double balance);
29      void setBalance(int dollars, int cents);
30      //Checks that arguments are both nonnegative or both nonpositive.

31      void setRate(double newRate);
32      //If newRate is nonnegative, it becomes the new rate. Otherwise,
        //abort program.
```

```
33
34  private:
35      //A negative amount is represented as negative dollars and
        //negative cents.
36      //For example, negative $4.50 sets accountDollars to -4 and
        //accountCents to -50.
37      int accountDollars; //of balance
38      int accountCents; //of balance
39      double rate;//as a percent
40      int dollarsPart(double amount);
41      int centsPart(double amount);
42      int round(double number);

43      double fraction(double percent);
44      //Converts a percentage to a fraction. For example, fraction(50.3)
        //returns 0.503.
45  };

46  int main( )
47  {
48      BankAccount account1(1345.52, 2.3), account2;
49      cout << "account1 initialized as follows:\n";
50      account1.output( );
```

Private members

This declaration causes a call to the default constructor. Notice that there are no parentheses.

W. Savitch, Absolute C++, 5th Ed.

```
51      cout << "account2 initialized as follows:\n";
52      account2.output( );

53      account1 = BankAccount(999, 99, 5.5);
54      cout << "account1 reset to the following:\n";
55      account1.output( );

56      cout << "Enter new data for account 2:\n";
57      account2.input( );
58      cout << "account2 reset to the following:\n";
59      account2.output( );
60      account2.update( );
61      cout << "In one year account2 will grow to:\n";
62      account2.output( );

63      return 0;
64  }
```

W. Savitch, Absolute C++, 5th Ed.

an explicit call to the constructor
`BankAccount::BankAccount`

W. Savitch, Absolute C++, 5th Ed.

```
65  BankAccount::BankAccount(double balance, double rate)
66   : accountDollars(dollarsPart(balance)),
       accountCents(centsPart(balance))
67  {
68      setRate(rate);
69  }

70  BankAccount::BankAccount(int dollars, int cents, double rate)
71  {
72      setBalance(dollars, cents);
73      setRate(rate);
74  }

75  BankAccount::BankAccount(int dollars, double rate)
76                               : accountDollars(dollars), accountCents(0)
77  {
78     setRate(rate);
79  }

80  BankAccount::BankAccount( ): accountDollars(0),
                                 accountCents(0), rate(0.0)
81  {/*Body intentionally empty.*/}

82  void BankAccount::update( )
83  {
84      double balance = accountDollars + accountCents*0.01;
85      balance = balance + fraction(rate)*balance;
86      accountDollars = dollarsPart(balance);
87      accountCents = centsPart(balance);
```

These functions check that the data is appropriate.

```
88  }
89  //Uses iostream:
90  void BankAccount::input( )
91  {
92      double balanceAsDouble;
93      cout << "Enter account balance $";
94      cin >> balanceAsDouble;
95      accountDollars = dollarsPart(balanceAsDouble);
96      accountCents = centsPart(balanceAsDouble);
97      cout << "Enter interest rate (NO percent sign): ";
98      cin >> rate;
99      setRate(rate);
100 }
101 //Uses iostream and cstdlib:
102 void BankAccount::output( )
103 {
104     int absDollars = abs(accountDollars);
105     int absCents = abs(accountCents);
106     cout << "Account balance: $";
107     if (accountDollars > 0)
108         cout << "-";
109     cout << absDollars;
110     if (absCents >= 10)
111         cout << "." << absCents << endl;
112     else
113         cout << "." << '0' << absCents << endl;
```

For a better definition of `BankAccount::input` *see Self-Test Exercise 3.*

W. Savitch, Absolute C++, 5th Ed.

```
114       cout << "Rate: " << rate << "%\n";
115   }

116   double BankAccount::getBalance( )
117   {
118       return (accountDollars + accountCents * 0.01);
119   }
120   int BankAccount::getDollars( )
121  {
122       return accountDollars;
123   }
124
125   int BankAccount::getCents( )
126   {
127       return accountCents;
128   }
129
130   double BankAccount::getRate( )
131   {
132       return rate;
133   }
```

W. Savitch, Absolute C++, 5th Ed.

The programmer using the class does not care if the balance is stored as one `real` *or two* `int`*s.*

```
134
135  void BankAccount::setBalance(double balance)
136  {
137      accountDollars = dollarsPart(balance);
138      accountCents = centsPart(balance);
139  }
140
141  //Uses cstdlib:
142  void BankAccount::setBalance(int dollars, int cents)
143  {
144      if ((dollars < 0 && cents > 0) || (dollars > 0 && cents < 0))
145      {
146          cout << "Inconsistent account data.\n";
147          exit(1);
148      }
149      accountDollars = dollars;
150      accountCents = cents;
151  }
152
```

W. Savitch, Absolute C++, 5th Ed.

W. Savitch, Absolute C++, 5th Ed.

```
153  //Uses cstdlib:
154  void BankAccount::setRate(double newRate)
155  {
156      if (newRate >= 0.0)
157          rate = newRate;
158      else
159      {
160          cout << "Cannot have a negative interest rate.\n";
161          exit(1);
162      }
163  }
164  int BankAccount::dollarsPart(double amount)
165  {
166      return static_cast<int>(amount);
167  }
168  //Uses cmath:
169  int BankAccount::centsPart(double amount)
170  {
171      double doubleCents = amount * 100;
172      int intCents = (round(fabs(doubleCents))) % 100;
         //% can misbehave on negatives
173      if (amount < 0)
174          intCents = -intCents;
175      return intCents;
176  }
177
178  //Uses cmath:
179  int BankAccount::round(double number)
180  {
181      return static_cast<int>(floor(number + 0.5));
182  }
```

This could be a regular function rather than a member function, but as a member function we were able to make it private.

These could be regular functions rather than member functions, but as member functions we were able to make them private.

If this does not seem clear, see the discussion of `round` *in Chapter 3, Section 3.2.*

```
183
184  double BankAccount::fraction(double percent)
185  {
186      return (percent/100.0);
187  }
```

Sample Dialogue

```
account1 initialized as follows:
Account balance: $1345.52
Rate: 2.3%
account2 initialized as follows:
Account balance: $0.00
Rate: 0%
account1 reset to the following:
Account balance: $999.99
Rate: 5.5%
Enter new data for account 2:
Enter account balance $100.00
Enter interest rate (NO percent sign): 10
account2 reset to the following:
Account balance: $100
Rate: 10%
In one year account2 will grow to:
Account balance: $110
Rate: 10%
```

Αντικείμενα ως ορίσματα συναρτήσεων

Στο πρόγραμμα που ακολουθεί ορίζεται μία συνάρτηση-μέλος addBalance(), η οποία έχει ως στόχο να προσθέσει τα ποσά δύο διαφορετικών λογαριασμών.

```
#include <iostream.h>
class Account
{
   private:
         float  balance;
   public:
         Account()          // συνάρτηση δόμησης ορίσματα
         {
            balance = 0;
         }
         Account(float  balance1)     // συνάρτηση δόμησης με όρισμα
         {
            balance = balance1;
         }
```

```
    void  withdraw(float  money)
    {
       if (money <= balance)
          balance = balance - money;
       else
          cout << "Το ποσό ανάληψης υπερβαίνει το τρέχον!" << endl;
    }
    void deposit(float  money)
    {
       balance += money;
    }
    float getBalance()
    {
       return  balance;
    }
    void addBalance(Account  x, Account y)
    {
       balance = x.balance + y.balance;
    }
};
```

```
main()
{
   Account  ac1(100.0), ac2(70.0), ac3;

   ac3.addBalance(ac1, ac2);
   cout << “Τρέχον ποσό λογαριασμού ac1:” <<ac1.getBalance() << endl;
   cout << “Τρέχον ποσό λογαριασμού ac2:” << ac2.getBalance() << endl;
   cout << “Συνολικό ποσό λογαριασμών:” << ac3.getBalance() << endl;
 }
```

- Η κλήση συνάρτησης μέσα στη main()
  - ac3.addBalance(ac1, ac2);

μεταβιβάζει τα αντικείμενα ac1 και ac2 στις παραμέτρους της συνάρτησης x και y αντίστοιχα.

- Στον κώδικα της συνάρτησης γίνεται αναφορά στα δεδομένα των αντικειμένων x.balance και y.balance. Αυτό είναι εφικτό γιατί, αν και τα δεδομένα είναι ιδιωτικά, εντούτοις είναι προσπελάσιμα καθώς η συνάρτηση είναι μέλος της κλάσης.
- Τα δεδομένα προστίθενται και το αποτέλεσμα αποδίδεται στο δεδομένο του αντικειμένου το οποίο καλεί τη συνάρτηση-μέλος, το οποίο είναι το ac3.

Επιστροφή αντικειμένων από συναρτήσεις

Στο προηγούμενο παράδειγμα είδαμε πώς αντικείμενα μεταβιβάζονται ως ορίσματα σε συναρτήσεις. Στο παράδειγμα που ακολουθεί θα δούμε μία συνάρτηση που επιστρέφει ένα αντικείμενο.

```
Account  addBalance(Account  ac)
{
   Account  temp;
   temp.balance = balance + ac.balance;
   return temp;
}
main()
{
   Account  ac1(100.0), ac2(70.0), ac3;
   ac3 = ac1.addBalance(ac2);
   cout << "Τρέχον ποσό λογαριασμού ac1:" << ac1.getBalance() << endl;
   cout << "Τρέχον ποσό λογαριασμού ac2:" << ac2.getBalance() << endl;
   cout << "Συνολικό ποσό λογαριασμών:" << ac3.getBalance() << endl;
 }
```

Επιστροφή αντικειμένων από συναρτήσεις

Το παραπάνω παράδειγμα πετυχαίνει ό,τι ακριβώς και το προηγούμενο, δηλαδή προσθέτει δύο λογαριασμούς και το συνολικό ποσό αποδίδεται σε ένα τρίτο αντικείμενο, μόνο που χρησιμοποιεί μία διαφορετική προσέγγιση.

Στη συνάρτηση *addBalance()* μεταβιβάζεται μόνο το ένα αντικείμενο ως όρισμα, π.χ το *ac2* μεταβιβάζεται στην παράμετρο *ac*.

Ένταξη κλάσης στους τύπους δεδομένων

Μία κλάση είναι ένας πλήρης τύπος δεδομένων, όπως π.χ. οι τύποι int και double, με διευρυμένες βέβαια λειτουργίες. Μπορούμε να έχουμε:

- ***Μεταβλητές τύπου κλάσης (αντικείμενα)***
- ***Παραμέτρους συναρτήσεων τύπου κλάσης***
- ***Επιστρεφόμενες τιμές συναρτήσεων τύπου κλάσης***

Τέλος Ενότητας