ΤΕΙ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΑΝΟΙΚΤΑ ΑΚΑΔΗΜΑΙΚΑ ΜΑΘΗΜΑΤΑ

ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ(Θ)

Ενότητα 3: 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 3

ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

ΔΙΔΑΣΚΩΝ: ΠΑΡΙΣ ΜΑΣΤΟΡΟΚΩΣΤΑΣ

Περιεχόμενα ενότητας

1. Υπερφόρτωση τελεστών (operator overloading)
2. Υπερφόρτωση αριθμητικών τελεστών
3. Υπερφόρτωση τελεστών σύγκρισης
4. Υπερφόρτωση τελεστών με χρήση συναρτήσεων-μελών
5. Υπερφόρτωση μοναδιαίων τελεστών
6. Υπερφόρτωση του τελεστή ανάθεσης
7. Υπερφόρτωση του τελεστή κλήσης συνάρτησης
8. Πίνακες
9. Πίνακες σε συναρτήσεις
10. Πίνακες ως δεδομένα κλάσεων
11. Πίνακες αντικειμένων

Σκοποί ενότητας

Υπερφόρτωση τελεστών (operator overloading)

Η υπερφόρτωση τελεστών είναι ένα από τα πιο ενδιαφέροντα χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού.

Αναφέρεται στην πρόσθετη χρήση των συνηθισμένων τελεστών, όπως π.χ. `+`, `-`, `*`, `>`, `<`, `==`, σε τύπους δεδομένων οριζόμενους από τον χρήστη.

Έως τώρα προτάσεις όπως

```
a = b + c
```

έχουμε δει να χρησιμοποιούνται μόνο σε βασικούς τύπους δεδομένων. Με την υπερφόρτωση τελεστών θα δούμε πώς μπορούμε να εκτελέσουμε τέτοιες προτάσεις, όταν τα `a`, `b` και `c` είναι αντικείμενα μίας κλάσης.

Υπερφόρτωση αριθμητικών τελεστών

Σε προηγούμενο παράδειγμα παρουσιάσθηκε τρόπος με τον οποίο δύο αντικείμενα της κλάσης Account μπορούν να προστεθούν, χρησιμοποιώντας μία συνάρτηση-μέλος:

```
ac3 = ac1.addBalance(ac2);
```

Εάν χρησιμοποιηθεί ο τελεστής + με υπερφόρτωση, τότε μπορούμε να έχουμε την εξής πρόταση:

```
ac3 = ac1 + ac2;
```

Στο πρόγραμμα που ακολουθεί υλοποιείται αυτή η περίπτωση:

Υπερφόρτωση αριθμητικών τελεστών

```
class Account
{
private:
    float  balance;
public:
    Account()
    {
        balance = 0;
    }
    Account(float  balance1)
    {
        balance = balance1;
    }
```

Υπερφόρτωση αριθμητικών τελεστών

```
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;
}
```

Υπερφόρτωση αριθμητικών τελεστών

```
   Account  operator + (Account  ac)
   {
         Account  temp;
         temp.balance = balance + ac.balance;
         return temp;
   }
};
main()
{
    Account  ac1(100.0), ac2(70.0), ac3;
   ac3 = 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;
 }
```

Υπερφόρτωση αριθμητικών τελεστών

Για να «αναγκάσουμε» τον τελεστή + να ενεργήσει πάνω σε ένα αντικείμενο, χρησιμοποιούμε τη δεσμευμένη λέξη operator. Συγκεκριμένα, γράφουμε μία συνάρτηση *όπου ο τύπος επιστρεφόμενης τιμής μπαίνει πρώτος, ακολουθούμενος από τη δεσμευμένη λέξη operator και αμέσως μετά τον ίδιο τον τελεστή*. Τέλος, μέσα σε παρενθέσεις γράφουμε τη λίστα των ορισμάτων.

Όταν στην καλούσα συνάρτηση εκτελείται η πρόταση:

```
ac3 = ac1 + ac2;
```

τότε γίνεται υπερφόρτωση του τελεστή + (επειδή τα ac1 και ac2 έχουν ορισθεί ως αντικείμενα), προστίθενται τα αντικείμενα ac1 και ac2 και το αποτέλεσμα αποδίδεται στο αντικείμενο ac3.

Θα πρέπει να διευκρινισθεί ότι η συνάρτηση χρησιμοποιεί ως όρισμα το αντικείμενο που βρίσκεται δεξιά του τελεστή (π.χ. το ac2). Ακόμα, η *συνάρτηση είναι μέλος στο αντικείμενο που βρίσκεται αριστερά του τελεστή* (π.χ. στο ac1) και έτσι η αναφορά στα δεδομένα αυτού του αντικειμένου είναι άμεση.

Υπερφόρτωση τελεστών σύγκρισης

Με παρόμοιο τρόπο μπορούμε να υλοποιήσουμε την υπερφόρτωση τελεστών σύγκρισης.

Στο επόμενο πρόγραμμα θα χρησιμοποιήσουμε τον τελεστή > (μεγαλύτερο από) με υπερφόρτωση, στην κλάση Account, για να μπορούμε να συγκρίνουμε δύο λογαριασμούς:

```
class Account
{
private:
        float  balance;
public:
        Account()
        {
                balance = 0;
        }
```

Υπερφόρτωση τελεστών σύγκρισης

```
Account(float  balance1)
{
        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;
}
```

Υπερφόρτωση τελεστών σύγκρισης

```
    float getBalance()
    {
        return  balance;
    }

    bool  operator > (Account  ac)
    {
        if (balance > ac.balance)  return true;
        else    return false;
    }
};    // τέλος της κλάσης
```

Υπερφόρτωση τελεστών σύγκρισης

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

   if (ac1 > ac2)
      cout << “Το ποσό του λογαριασμού ac1 είναι μεγαλύτερο.”
           << endl;
   else
      cout << “Το ποσό του ac2 είναι μεγαλύτερο ή είναι ίσοι.”
           << endl;
}
```

Τελεστές που δεν επιδέχονται υπερφόρτωση

- Οι τελεστές
  - :: (διάκρισης εμβέλειας)
  - . (πρόσβασης σε μέλος)
  - .* (πρόσβασης σε μέλος μέσω δείκτη σε μέλος)
  - ?: (υποθετικός τελεστής)

Δεν μπορούν να υπερφορτωθούν.

- Δεν μπορούν να δημιουργηθούν νέοι τελεστές της μορφής ** ή &|.

Υπερφόρτωση τελεστών με χρήση συναρτήσεων-μελών

Θεωρούμε την ακόλουθη κλάση:

```
class rectangle {
private:
        float side_a, side_b;
public:
        int color;
        rectangle(float a, float b);
        rectangle();
        void set_sides(float a, float b);
        float area();
        void show();
};
```

Υπερφόρτωση τελεστών με χρήση συναρτήσεων-μελών

```
void rectangle :: set_sides(float a, float b)
{
        side_a = a;       side_b = b;
}
float rectangle :: area()
{
        return side_a * side_b;
}
void rectangle :: show()
{
        cout << .... << endl;
}
```

Υπερφόρτωση τελεστών με χρήση συναρτήσεων-μελών

```
rectangle :: rectangle(float a, float b)
{
        side_a = a;
        side_b = b;
        color = 0;
}
bool rectangle :: operator > (rectangle op2)
{
        if (this -> area() > op2.area())  return true;
        else      return false;
}
```

Εννοείται:

(area() > op2.area())

Υπερφόρτωση τελεστών με χρήση συναρτήσεων-μελών

Κατά συνέπεια, το σχετικό τμήμα κώδικα της **main** είναι:

```
rectangle rec1(10,2), rec2(5,7);
if (rec2 > rec1)
        cout << “rec2 > rec1” << endl;
```

Αποτέλεσμα:

rec2 > rec1

Υπερφόρτωση μοναδιαίων τελεστών

Σε περίπτωση που υπάρχει μεταθεματική και προθεματική σημειογραφία (postfix/prefix) θα πρέπει να έχουμε δύο εκδοχές, μία για κάθε σημειογραφία.

(α) Προθεματική σημειογραφία:

```
rectangle rectangle :: operator ++()
{
        this -> side_a ++;     ή     side_a ++;
        this -> side_b ++;     ή     side_b ++;
        return *this;
}
```

Δεν υπάρχει παράμετρος.

Υπερφόρτωση μοναδιαίων τελεστών

(β) Στη μεταθεματική σημειογραφία υπάρχει παράμετρος που δε χρησιμοποιείται, γι' αυτό και δεν αναφέρεται το όνομά της:

```
rectangle rectangle :: operator ++(int)
{
        rectangle temp = *this;
        this -> side_a ++;
        this -> side_b ++;
        return temp;
}
```

Όπως διαπιστώνεται, οι τρέχουσες τιμές του αντικειμένου αντιγράφονται στο τοπικό αντικείμενο temp, το οποίο θα επιστραφεί στην καλούσα συνάρτηση, έτσι ώστε οι αυξήσεις να μην επηρεάσουν την παράσταση στην οποία εμπλέκεται το αντικείμενο.

Υπερφόρτωση μοναδιαίων τελεστών

Παράδειγμα (τμήμα της main):

```
rectangle rec1(10,2), rec2(5,7), rec3;
cout << “rec1 = ”;
rec1.show();
rec1++;
cout << “rec1 = ”;
rec1.show();
rec3 = rec2 ++;
cout << “rec3 = ”;
rec3.show();
cout << “rec2 = ”;
rec2.show();
```

Αποτελέσματα:

rec1 = 10x2, color = 0

rec1 = 11x3, color = 0

rec3 = 5x7, color = 0

rec2 = 6x8, color = 0

Αντιγράφονται οι «παλιές» διαστάσεις του rec2, οι οποίες ακολούθως αυξάνονται κατά 1.

Υπερφόρτωση του τελεστή ανάθεσης

Όταν χρησιμοποιείται η δυναμική διαχείριση μνήμης προτιμάται η υπερφόρτωση.

Παράδειγμα εφαρμογής: Αντιγραφή μέρους των μεταβλητών-μελών και όχι του συνόλου, π.χ. Αντιγραφή μόνο των πλευρών και όχι του χρώματος:

```
rectangle rectangle :: operator =(rectangle op2)
{
        this -> side_a = op2.side_a;
        this -> side_b = op2.side_b;
        return *this;
}
```

Υπερφόρτωση του τελεστή ανάθεσης

Παράδειγμα (τμήμα της main):

```
rectangle rec1(10,2), rec2(5,7);
rec1.color = 1;
rec2.color = 2;
cout << “rec1 = ”;
rec1.show();
rec1 = rec2;
cout << “rec1 = ”;
rec1.show();
```

Αποτελέσματα:

rec1 = 10x2, color = 1

rec1 = 5x7, color = 1

Δηλαδή το χρώμα δεν άλλαξε.

Υπερφόρτωση του τελεστή κλήσης συνάρτησης

‘Οταν γίνεται αναφορά στα αντικείμενα μίας κλάσης σε μορφή συναρτήσεων π.χ. rec(4,6), τα ορίσματα που ακολουθούν μεταβιβάζονται στη συνάρτηση υπερφόρτωσης του τελεστή ().

Παράδειγμα:

```
rectangle rectangle :: operator () (float a, float b)
{
        this -> side_a = a;
        this -> side_b = b;
        return *this;
}
```

Ουσιαστικά δρα ως συνάρτηση *ανάθεσης* τιμής ή *τροποποίησης* τιμής.

Υπερφόρτωση του τελεστή κλήσης συνάρτησης

Εάν ορίσουμε ένα μόνον όρισμα, μπορούμε π.χ. να κλιμακοποιήσουμε:

```
rectangle rectangle :: operator () (float scale)      {
        this -> side_a = side_a * scale;;
        this -> side_b = side_b * scale;
        return *this;
}
```

Παράδειγμα (τμήμα της main):

```
rectangle rec1(10,2);
rec1.color = 1;
cout << “rec1 = ”;        rec1.show();
rec1(5,9);
cout << “rec1 = ”;        rec1.show();
rec1(3);
cout << “rec1 = ”;        rec1.show();
```

Αποτελέσματα:

rec1 = 10x2, color = 1

rec1 = 5x9, color = 1

rec1 = 15x27, color = 1

Υπερφόρτωση τελεστών με χρήση συναρτήσεων-μελών

Οι φίλιες (friend) συναρτήσεις δηλώνονται στο public τμήμα μίας κλάσης με το πρόθεμα friend, αλλά ορίζονται όπως οι κανονικές συναρτήσεις εκτός της κλάσης, χωρίς το class_name ::, και προσπελαύνουν τα ιδιωτικά μέλη των αντικειμένων της κλάσης. Κατά συνέπεια δεν αποτελούν συναρτήσεις-μέλη και ο δείκτης this δε μεταβιβάζεται.

Παράδειγμα φίλιας συνάρτησης:

```
class rectangle
{
private:
        float side_a, side_b;
public:
        rectangle(float a, float b);
        rectangle();
```

Υπερφόρτωση τελεστών με χρήση συναρτήσεων-μελών

```
    float area();
    void set_sides(float a, float b);
    friend void to_zero(rectangle &rec);
};          // τέλος της κλάσης rectangle
void to_zero(rectangle &rec)        {
    rec.side_a = 0;
    rec_side_b = 0;
}
main     {
    rectangle rec1;
    rec1.set_sides(8,7);
    cout << “Area of rec1 (prior) = ” << rec1.area() << endl;
    to_zero(rec1);
    cout << “Area of rec1 (afterwards) = ” << rec1.area() << endl;
```

Δε συνδέεται με την κλάση μέσω ::

Αποτελέσματα:

Area of rec1 (prior) = 56

Area of rec1 (afterwards) = 0

Πίνακες

Οι πίνακες (συλλογές στοιχείων του ίδιου τύπου δεδομένων) κατατάσσονται στις στατικές δομές δεδομένων.

Με τον όρο *στατική δομή δεδομένων* εννοείται ότι το ακριβές μέγεθος της απαιτούμενης κύριας μνήμης καθορίζεται κατά τη στιγμή του προγραμματισμού τους, επομένως κατά τη στιγμή της μετάφρασης του προγράμματος και όχι κατά τη στιγμή της εκτέλεσής του.

Ένα άλλο χαρακτηριστικό των πινάκων είναι ότι τα στοιχεία τους αποθηκεύονται σε συνεχόμενες θέσεις μνήμης.

Η δήλωση των στοιχείων ενός πίνακα και η μέθοδος αναφοράς τους εξαρτάται από τη συγκεκριμένη γλώσσα υψηλού επιπέδου που χρησιμοποιείται. Όμως γενικά, η αναφορά στα στοιχεία ενός πίνακα γίνεται με τη χρήση του συμβολικού ονόματος του πίνακα ακολουθούμενου από την τιμή ενός ή περισσοτέρων αριθμοδεικτών (indices) σε παρένθεση ή αγκύλη.

Πίνακες

Ένας πίνακας μπορεί να είναι μονοδιάστατος, δισδιάστατος, τρισδιάστατος και γενικά ν-διάστατος πίνακας.

Στη C++ επιτρέπεται ο ορισμός πινάκων οποιασδήποτε διάστασης, αλλά πέραν της δεύτερης, η χρήση τους ενέχει τέτοια ποικιλία προβλημάτων ώστε να είναι αποτρεπτική. Για το λόγο αυτό, κυρίως θα ασχοληθούμε με μονοδιάστατους (*one-dimensional*) και δισδιάστατους (*two-dimensional*) πίνακες.

Πίνακες

Ορισμός μονοδιάστατου πίνακα ακεραίων:

```
#define N 10

main()
{
   int  p[N];
   .
   .
   .
   .
}
```

Ορισμός δισδιάστατου πίνακα ακεραίων:

```
#define M 8
#define N 10

main()
{
   int  p[M][N];
   .
   .
   .
   .
}
```

Πίνακες σε συναρτήσεις

- **Στατικοί τοπικοί πίνακες**
  - Διατηρούν τις τιμές τους μεταξύ των κλήσεων μιας συνάρτησης
  - Οι στατικοί πίνακες αρχικοποιούνται με μηδέν.

```
static int array[3];
```

- **Αυτόματοι τοπικοί πίνακες**
  - Δημιουργούνται (και καταστρέφονται) με κάθε κλήση της συνάρτησης.

Πίνακες σε συναρτήσεις

```
1   // ΠΑράδειγμα στατικών και μη πινάκων
2   // Οι στατικοί πίνακες αρχικοποιούνται με μηδέν
3   #include <iostream>
4
5   using namespace std;
6   using std::endl;
7
8   void staticArrayInit( void );       // δήλωση συνάρτησης
9   void automaticArrayInit( void );    // δήλωση συνάρτησης
10
11  int main()
12  {
13     cout << "First call to each function:\n";
14     staticArrayInit();
15     automaticArrayInit();
16
17     cout << "\n\nSecond call to each function:\n";
18     staticArrayInit();
19     automaticArrayInit();
20     cout << endl;
21
22     return 0;
23
24  }
25
```

Πίνακες σε συναρτήσεις

```
26
27  void staticArrayInit( void )
28  {
29
30     static int array1[ 3 ];
31
32     cout << "\nValues on entering staticArrayInit:\n";
33
34
35     for ( int i = 0; i < 3; i++ )
36        cout << "array1[" << i << "] = " << array1[ i ] << "   ";
37
38     cout << "\nValues on exiting staticArr
39
40
41     for ( int j = 0; j < 3; j++ )
42        cout << "array1[" << j << "] = "
43             << ( array1[ j ] += 5 ) << "   ";
44
45  }
46
```

Στατικός πίνακας, αρχικοποιείται με μηδενικά στην πρώτη κλήση της συνάρτησης.

Τα δεδομένα μεταβλήθηκαν και οι μεταβολές θα διατηρηθούν και μετά το πέρας της συνάρτησης.

Πίνακες σε συναρτήσεις

```
47
48 void automaticArrayInit( void )
49 {
50
51    int array2[ 3 ] = { 1, 2, 3 };
52
53    cout << "\n\nValues on entering automaticArrayInit:\n";
54
55
56    for ( int i = 0; i < 3; i++ )
57       cout << "array2[" << i << "] = " << array2[ i ] << "   ";
58
59    cout << "\nValues on exiting automaticA
60
61
62    for ( int j = 0; j < 3; j++ )
63       cout << "array2[" << j << "] = "
64            << ( array2[ j ] += 5 ) << "   ";
65
66 }
```

Αυτόματος πίνακας, δημιουργείται εκ νέου με κάθε κλήση συνάρτησης.

Αν και ο πίνακας μεταβλήθηκε, θα καταστραφεί με το πέρας της συνάρτησης και οι μεταβολές θα χαθούν.

Πίνακες σε συναρτήσεις

```
First call to each function:

Values on entering staticArrayInit:
array1[0] = 0  array1[1] = 0  array1[2] = 0
Values on exiting staticArrayInit:
array1[0] = 5  array1[1] = 5  array1[2] = 5

Values on entering automaticArrayInit:
array2[0] = 1  array2[1] = 2  array2[2] = 3
Values on exiting automaticArrayInit:
array2[0] = 6  array2[1] = 7  array2[2] = 8

Second call to each function:

Values on entering staticArrayInit:
array1[0] = 5  array1[1] = 5  array1[2] = 5
Values on exiting staticArrayInit:
array1[0] = 10  array1[1] = 10  array1[2] = 10

Values on entering automaticArrayInit:
array2[0] = 1  array2[1] = 2  array2[2] = 3
Values on exiting automaticArrayInit:
array2[0] = 6  array2[1] = 7  array2[2] = 8
```

Πίνακες ως δεδομένα κλάσεων

Οι πίνακες μπορούν να χρησιμοποιηθούν ως στοιχεία δεδομένων στις κλάσεις.

Ας εξετάσουμε ένα παράδειγμα, όπου ορίζεται μία κλάση με το όνομα Student και δεδομένα τον αριθμό μητρώου ενός σπουδαστή και έναν πίνακα, όπου αποθηκεύονται 6 βαθμολογίες του για 6 αντίστοιχα μαθήματα.

```
#include <cstdlib>
#include <iostream>
#include <iomanip>
using namespace std;
const int N = 6;
```

theory_3_matrices_1.cpp

Πίνακες ως δεδομένα κλάσεων

```
class Student
{
private:
        int  am;
        float  vath[N];
public:
        Student();
        void  readData();
        float  getAvg();
        float  getMin();
        float  getMax();
};
```

Πίνακες ως δεδομένα κλάσεων

```
main()
{
        Student  s1;
        s1.readData();
        cout << "\tGPA (Grade Point Average) = “
                << fixed << setprecision(2) << s1.getAvg() << endl;
        cout << "\tMaximum grade = “
                << setprecision(2) << s1.getMax() << endl;
        cout << "\tMinimum grade = “
                << setprecision(2) << s1.getMin() << endl;

        cout << endl << endl;
        system("PAUSE");
}
```

Πίνακες ως δεδομένα κλάσεων

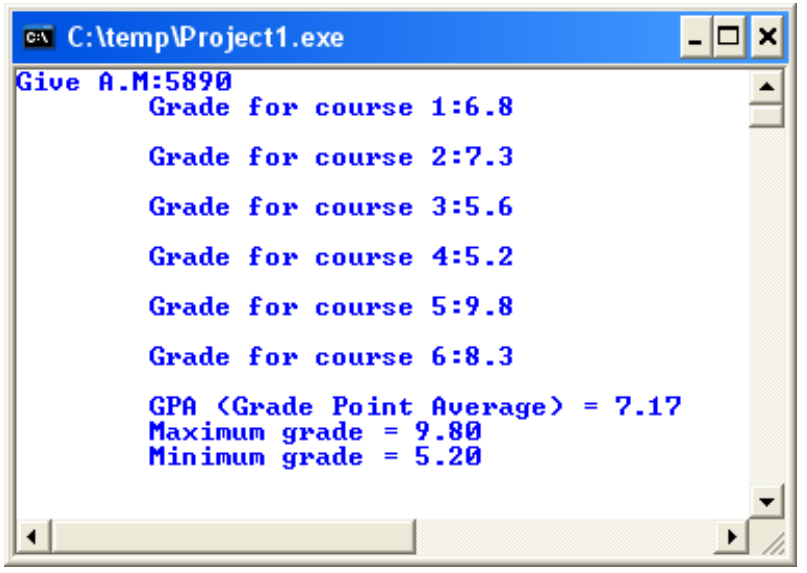
```
Student :: Student()
{
        int i;
        am = 0;
        for (i=0; i<N; i++)          vath[i] = 0;
}
void Student :: readData()
{
        int i;
        cout << "Give A.M:";        cin >> am;
        for (i=0; i<N; i++)
        {
                cout << "\tGrade for course " << i+1 << ":";
                cin >> vath[i];    cout << endl;
        }
 }
```

Πίνακες ως δεδομένα κλάσεων

```
float Student :: getAvg()
{
        int i;
        float  sum = 0, avg;
        for (i=0; i<N; i++)   sum += vath[i];
        avg = sum/N;
        return avg;
}
float Student :: getMin()
{
        int i;
        float minim;
        minim = vath[0];
        for (i=1; i<N; i++)  if (vath[i] < minim)  minim = vath[i];
   return minim;
 }
```

Πίνακες ως δεδομένα κλάσεων

```
float Student :: getMax()
{
        int i;
        float  maxim;
        maxim = vath[0];
        for (i=1; i<N; i++)
                if (vath[i] > maxim)  maxim = vath[i];
        return maxim;
 }
```



Πίνακες αντικειμένων

Όπως ένα αντικείμενο μπορεί να περιέχει έναν πίνακα, μπορεί να ισχύει και το αντίστροφο, δηλαδή να έχουμε έναν πίνακα που να περιέχει αντικείμενα.

```
#include <cstdlib>
#include <iostream>

using namespace std;

const int N = 4;
class Employee
{
private:
        int rn;
        char name[20];
        float salary;
```

theory_3_matrices_2.cpp

Πίνακες αντικειμένων

```
public:
  void readData();
  void printData();
  int getRN();
};          // τέλος της κλάσης

main()
{
  Employee emp[N];
  int  i, armit;
  bool  found = false;
  cout << "Provide details for " << N << " employees:" << endl;
  for (i=0; i<N; i++)
  {
          cout << endl << "\tEmployee " << i+1 << ":" << endl;
          emp[i].readData();
  }
```

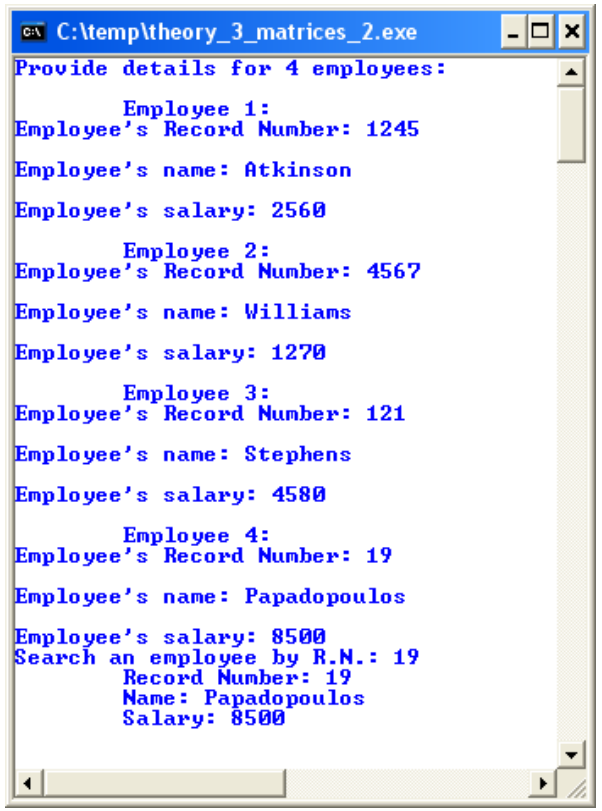
Πίνακες αντικειμένων

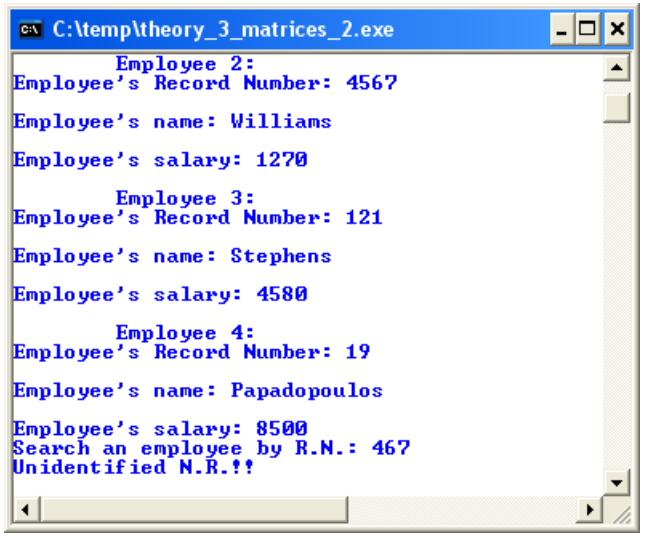
```
        cout << "Search an employee by R.N.: ";
        cin >> armit;
        i = 0;
        found = false;
        while  (i<N  &&  found == false)
        {
                if (emp[i].getRN() == armit)          found = true;
                else       i++;
          }
          if (found == true)          emp[i].printData();
          else                cout << "Unidentified N.R.!!" << endl;

        cout << endl << endl;
        system("PAUSE");
}       // τέλος της main
```

Πίνακες αντικειμένων

```
void Employee :: readData()
{
        cout << "Employee's Record Number: ";
        cin >> rn;
        cout << endl << "Employee's name: ";
        cin >> name;
        cout << endl << "Employee's salary: ";
        cin >> salary;
}
void Employee :: printData()
{
        cout << "\tRecord Number: " << rn << endl;
        cout << "\tName: " << name << endl;
        cout << "\tSalary: " << salary << endl;
}
```

```
int Employee :: getRN()
{
        return rn;
}
```





Πίνακες αντικειμένων

Στο προηγούμενο πρόγραμμα ορίζεται ένας πίνακας αντικειμένων **emp** τύπου **Employee**.

Κατόπιν, με μία δομή επανάληψης **for** πληκτρολογούνται πληροφορίες για κάθε ένα από τα 4 αντικείμενα του πίνακα, με την κλήση της συνάρτησης ***readData()***.

Τέλος, πληκτρολογείται ένας τυχαίος αριθμός μητρώου και αναζητείται στον πίνακα αντικειμένων. Εάν υπάρχει, εμφανίζονται οι πληροφορίες για τον συγκεκριμένο εργαζόμενο.

Τέλος Ενότητας