project

4

Φοιτητές και Μαθήματα Αλλιώς

Περιεχόμενα:

Prj04.1 Το Πρόβλημα

Το πρόβλημα που έχουμε να λύσουμε είναι μια παραλλαγή του προβλήματος που λύσαμε στο Project 3. Οι επιπλέον απαιτήσεις είναι:

Α. Σε κάθε αντικείμενο κλάσης Student θα αποθηκεύονται –σε δυναμικό πίνακα– οι κωδικοί μαθημάτων στα οποία έχει εγγραφεί ο φοιτητής.

Αυτό θα μας δημιουργήσει το εξής πρόβλημα: Όταν φυλάγουμε τα αντικείμενα τύπου *Student* στο `students.dta` αυτά θα έχουν διαφορετικό μέγεθος. Έτσι, δεν θα μπορούμε να χειριστούμε το αρχείο με τον τρόπο που μάθαμε στην §15.13. Για να ανακτήσουμε τη δυνα-

τοτητα κατ' ευθείαν πρόσβασης στα αντικείμενα του αρχείου θα χρειαστούμε ένα **ευρετήριο** (index) με στοιχεία τύπου

```
struct IndexEntry
{
   unsigned int sIdNum;
   size_t       loc;
}; // IndexEntry
```

Το πρόγραμμά μας θα πρέπει να οργανώνει αυτά τα στοιχεία σε έναν πίνακα index *που θα τον φυλάγει σε ένα (μη-μορφοποιημένο) αρχείο* `students.ndx`.

Β. Εκτός από το πρόγραμμα που γράψαμε –και θα ξαναγράψουμε με κάποιες διαφορές– και το ονομάζουμε ***πρόγραμμα δημιουργίας****, θέλουμε άλλο ένα πρόγραμμα που θα το ονομάσουμε* ***πρόγραμμα εκμετάλλευσης****.*

Αυτό το πρόγραμμα θα κάνει κατά βάση μια δουλειά: θα δέχεται από τον χρήστη έναν αριθμό μητρώου φοιτητή/τριας και θα δείχνει τα στοιχεία του/της όπως υπάρχουν στο `students.dta`*. Αν δεν υπάρχει τέτοιος αριθμό μητρώου θα δίνει κατάλληλο μήνυμα. Αυτό θα επαναλαμβάνεται μέχρι να ζητηθεί "ΤΕΛΟΣ" από τον χρήστη.*

Ακόμη, αυτή τη φορά:

- θα γράψουμε τις κλάσεις μας με βάση τη «συνταγή» της §21.8,
- θα χειριστούμε με πιο «ορθόδοξο» τρόπο τους εξωτερικούς πίνακες μαθημάτων και φοιτητών και
- θα γράψουμε κάπως διαφορετικά τις κλάσεις εξαιρέσεων: κάθε αντικείμενο-εξαίρεση θα περιέχει και το κλειδί του αντικειμένου που έχει το πρόβλημα. Το είχαμε υποσχεθεί στην §19.4.

Σε σχέση με το πρώτο σημείο θα πρέπει να τονίσουμε το εξής: *Ο σωστός τρόπος εργασίας είναι αυτός που είδαμε στο Project 3, δηλαδή βλέπουμε τις ανάγκες για την κάθε κλάση όπως υπαγορεύονται από τις εφαρμογές που τη χρησιμοποιούν και την εξοπλίζουμε καταλλήλως.* Η «συνταγή» της §21.8 ασχολείται με μερικές πολύ συνηθισμένες ανάγκες που καλό είναι να αντιμετωπίζονται με πάγιο τρόπο.

Prj04.2 Η Κλάση *Course*

Πριν αρχίσουμε να εφαρμόζουμε τη «συνταγή» μας για την κλάση *Course* θα πρέπει να πάρουμε υπόψη μας κάτι που υπάρχει στις προδιαγραφές και αφορα την κλάση *Student*: «*Σε κάθε αντικείμενο κλάσης Student θα αποθηκεύονται –σε δυναμικό πίνακα– οι κωδικοί μαθημάτων στα οποία έχει εγγραφεί ο φοιτητής.*»

Είναι φανερό ότι στον πίνακα αυτόν θα πρέπει να κάνουμε αναζητήσεις. Αν θέλουμε να χρησιμοποιήσουμε το περίγραμμα *linSearch()* θα πρέπει να έχουμε επιφορτώσει τον τελεστή "`!=`" για τον τύπο του *cCode*. Αυτό δεν μπορεί να γίνει αν έχουμε δηλώσει:

```
   char       cCode[cCodeSz];
```

Θα πρέπει να ορίσουμε έναν νέο τύπο (κλάση περιτυλίγματος), ας τον πούμε *CourseKey*, ως εξής:

```
class Course
{
public:
   enum { cCodeSz = 8 };
   struct CourseKey
   {
      char s[cCodeSz];
      explicit CourseKey( string aKey="" )
      {  strcpy( s, aKey.c_str(), cCodeSz-1 );  s[cCodeSz-1] = '\0';  }
   }; // CourseKey
// . . .
```

και να δηλώσουμε:

```
// . . .
private:
// . . .
   CourseKey    cCode;              // κωδικός μαθήματος
```

Τώρα μπορούμε να επιφορτώσουμε τον "!=" ως εξής:

```
bool operator!=( const Course::CourseKey& lhs, const Course::CourseKey& rhs )
{  return ( strcmp(lhs.s, rhs.s) != 0 );  }
```

Και τώρα ας εφαρμόσουμε τη συνταγή μας.

ΑΝ, αναλλοίωτη:

((*strlen*(*cCode*.*s*) == *cCodeSz*-1) || (*strlen*(*cCode*.*s*) == 0)) && (*strlen*(*cTitle*) < *cTitleSz*) &&
(1 <= *cFSem* <= 8) && (*cSector* ∈ {'Υ','Π','Μ','Γ','Ξ'})[1] &&
(*cCateg* ∈ {"ΜΓΥ","ΜΕΥ","ΜΕ","ΔΟΝ"})[2] && (*cWH* >= 0) && (*cUnits* >= 0) &&
((*strlen*(*cPrereq*.*s*) == *cCodeSz*-1) || (*strlen*(*cPrereq*.*s*) == 0)) &&
((*strlen*(*cPrereq*.*s*) == *cCodeSz*-1) ⇒ (*cPrereq* != *cCode*)) &&
(*cNoOfStudents* >= 0)

Παρατηρήσεις: ▶

1. Οι *strlen*(*cCode*.*s*) == *cCodeSz*-1 και *strlen*(*cTitle*) < *cTitleSz* είναι διαφορετικές: Ο κωδικός του μαθήματος πρέπει να έχει ακριβώς 7 χαρακτήρες (και άλλα χαρακτηριστικά δομής –που περιγράφονται στην άσκ. Prj03-1– αλλά εδώ τα αγνοούμε.) *Αν δοθούν λιγότεροι ή περισσότεροι θα πρέπει να ρίξουμε εξαίρεση.* Ο τίτλος του μαθήματος πρέπει να έχει λιγότερους από 80 χαρακτήρες. Αν μας δοθούν 80 ή περισσότεροι *δεν ρίχνουμε εξαίρεση*· απλώς κρατούμε τους πρώτους 79 (*cTitleSz*-1) και αγνοούμε τους παραπανήσιους.
2. Η *strlen*(*cCode*.*s*) == 0 θα πρέπει να επιτρέπεται στην περίπτωση δήλωσης της μορφής "Course c0" (και μόνον).
3. Οι *cWH* == 0 και *cUnits* == 0 είναι δεκτές μόνον στον δημιουργό και όχι στις αντίστοιχες "*set*".
4. Η

((*strlen*(*cPrereq*.*s*) == *cCodeSz*-1) || (*strlen*(*cPrereq*.*s*) == 0)) &&
((*strlen*(*cPrereq*.*s*) == *cCodeSz*-1) ⇒ (*cPrereq* != *cCode*))

δεν φτάνει: θα πρέπει επιπλέον να υπάρχει μάθημα με τέτοιον κωδικό στον πίνακα μαθημάτων. Για να μπορέσεις να βάλεις αυτόν τον περιορισμό στην αναλλοίωτη και τον αντίστοιχο έλεγχο στις μεθόδους θα πρέπει να σχεδιάσεις τις κλάσεις σου έτσι ώστε τα αντικείμενα τύπου *Course* να υπάρχουν μόνον μέσα σε κάποιον «πίνακα μαθημάτων».
5. Ο έλεγχος «αυτοαναφοράς» (κωδικός προαπαιτουμένου ίδιος με κωδικό μαθήματος) θα πρέπει να γίνεται όταν αλλάζουμε κωδικό προαπαιτουμένου αλλά και όταν αλλάζουμε κωδικό μαθήματος.◀

Η κλάση εξαιρέσεων θα είναι ως εξής:

```
struct CourseXptn
{
   enum { . . . };
   Course::CourseKey objKey;
   char              funcName[100];
   int               errorCode;
   char              errStrVal[100];
   int               errIntVal;
   CourseXptn( const Course::CourseKey& obk, const char* mn, int ec,
               const char* sv="" )
      : objKey( obk ),  errorCode( ec )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';
      strncpy( errStrVal, sv, 99 );  errStrVal[99] = '\0';  }
```

[1] Έχουν σχέση με το Τμήμα Βιομηχανικής Πληροφορικής του ΤΕΙ Καβάλας.

[2] Αναφέρονται στα προγράμματα σπουδών ΤΕΙ κάποιας εποχής...

```
}; // CourseXptn
```

Το νέο στοιχείο εδώ είναι το μέλος *objKey*. Εδώ θα βάζουμε το κλειδί του αντικειμένου που έχει το πρόβλημα.

Να σημειώσουμε ακόμη ότι για να μπορούμε να χρησιμοποιήσουμε στη δήλωση του *objKey* τη σταθερά *Course*`::`*cCodeSz* θα πρέπει να μεταφέρουμε τον ορισμό της σε περιοχή "`public`" στη δήλωση της *Course*.

ΔΕ, ερήμην δημιουργός**:** Θέλουμε να έχουμε δυνατότητα να κάνουμε δηλώσεις σαν τις:

```
   Course c0;
   Course c1( "EY01010" );
   Course c2( "EY01010", "Εισαγωγή στον Προγραμματισμό (0)" );
```

Σύμφωνα με αυτά που είπαμε στην §21.11.1, επειδή –όπως φαίνεται στο δεύτερο από τα παραδείγματα– ο δημιουργός θα επιτρέπει και τη μετατροπή τύπου (κλήση με μια και μόνη παράμετρο τύπου *string*), δηλώνουμε:

```
   explicit Course( string aCode="", string aTitle="" );
```

και ορίζουμε:

```
Course::Course( string aCode, string aTitle )
{
   if ( aCode.length() != cCodeSz-1 &&
        (aCode.length() != 0 || aTitle.length() != 0) )
      throw CourseXptn( CourseKey(""), "Course", CourseXptn::keyLen,
                        aCode.c_str() );
   cCode = CourseKey( aCode );
   strncpy( cTitle, aTitle.c_str(), cTitleSz-1 );  cTitle[cTitleSz-1] = '\0';
   cFSem = 0;
   cCompuls = false;
   cCateg[0] = '\0';
   cWH = 0;
   cUnits = 0;
   cPrereq = CourseKey( "" );
   cNoOfStudents = 0;
} // Course::Course
```

Να εξηγήσουμε τον έλεγχο που κάνουμε: Ο κωδικός μαθήματος θα πρέπει να έχει μήκος 7. Μπορεί να έχει και μήκος 0 αλλά μόνον σε δήλωση της μορφής "`Course c0`". Όμως δεν επιτρέπεται δήλωση της μορφής:

```
   Course c2( "", "Εισαγωγή στον Προγραμματισμό (0)" );
```

όπου έχουμε τίτλο μαθήματος αλλά όχι κωδικό. Δηλαδή, πιο σωστά, θα έπρεπε να έχουμε στην αναλλοίωτη:

(*strlen*(*cCode*`.`*s*) `==` *cCodeSz*-1) `||` (*strlen*(*cCode*`.`*s*) `==` 0 `&&` *strlen*(*cTitle*) `==` 0)

Η συνθήκη ελέγχου είναι άρνηση αυτής της συνθήκης για τις αντίστοιχες παραμέτρους του δημιουργού.

Αν μας δώσουν «παράνομο» κωδικό μαθήματος στον δημιουργό τι κλειδί θα βάλουμε στην εξαίρεση; Όπως βλεπεις, εδώ βάλαμε "`CourseKey("")`" (κενός ορμαθός). Η «αμαρτωλή» τιμή (*aCode*) περνάει με το τελευταίο όρισμα.

Κατά τα άλλα, αντί για "`cCode = CourseKey( aCode )`" θα μπορούσαμε να είχαμε γράψει:

```
   CourseKey cCode;
   strcpy( cCode.s, aCode.c_str() );
```

και αντί για "`cPrereq = CourseKey( "" )`" θα μπορούσαμε να έχουμε:

```
   cPrereq.s[0] = '\0';
```

Παρατήρηση: ▶

Μήπως έπρεπε να βάλουμε το "**explicit**" και στον δημιουργό της *CourseKey*; Όχι! Η τιμή που μας ενδιαφέρει είναι ακριβώς τύπου "`char[cCodeSz]`". Την «μεταμφιέσαμε» σε "**struct**" για τεχικούς λόγους.◀

ΔΑ, δημιουργός αντιγραφής: Τα αντικείμενα της κλάσης δεν δεσμεύουν *δυναμικώς* πόρους του συστήματος. Έτσι, δεν απαιτείται να γράψουμε δημιουργό αντιγραφής· αυτός που γράφεται αυτομάτως από τον μεταγλωττιστή είναι σωστός.
ΤΕ, τελεστής εκχώρησης: Για τον ίδιο λόγο δεν χρειάζεται να γράψουμε και τελεστή εκχώρησης.
ΚΑ, καταστροφέας: Ούτε καταστροφέα χρειάζεται να γράψουμε· εμείς όμως θα βάλουμε έναν "κενό":

```
   ~Course() { };
```

GE, συναρτήσεις "*get*":

```
   const char* getCode() const {  return cCode.s;  }
   const char* getTitle() const {  return cTitle;  }
   unsigned int getFSem() const {  return cFSem;  }
   bool getCompuls() const {  return cCompuls;  }
   char getSector() const {  return cSector;  }
   const char* geCateg() const {  return cCateg;  }
   unsigned int getWH() const {  return cWH;  }
   unsigned int getUnits() const {  return cUnits;  }
   unsigned int getNoOfStudents() const {  return cNoOfStudents; }
   const char* getPrereq() const {  return cPrereq.s;  }
```

Πρόσεξε την πρώτη και την τελευταία: Προτιμούμε να επιστρέψουμε τιμή τύπου "`const char*`" –δηλαδή τον «φυσικό» τύπο των μελών– και όχι `CourseKey`, που είναι «μεταμφιεσμένος».
SE, συναρτήσεις "*set*": Ξεκινούμε με τη *setCode* που τώρα έχει και ελέγχους:

```
void Course::setCode( string aCode )
{
   if ( aCode.length() != cCodeSz-1 )
      throw CourseXptn( CourseKey(cCode), "setCode", CourseXptn::keyLen,
                        aCode.c_str() );
   if ( (aCode.length() != 0) && (CourseKey(aCode) == cPrereq) )
      throw CourseXptn( CourseKey(cCode), "setCode", CourseXptn::autoRef,
                        aCode.c_str() );
   strcpy( cCode.s, aCode.c_str() );
} // Course::setCode
```

Πρόσεξε ότι

- Τώρα δεν δεχόμαστε κωδικούς που το μήκος τους δεν είναι 7.[3]
- Για να γίνει η σύγκριση "`CourseKey(aCode) == cPrereq`" θα πρέπει να έχουμε επιφορτώσει τον "`==`" για τον *CourseKey*:

```
bool operator==( const Course::CourseKey& lhs, const Course::CourseKey& rhs )
{  return ( !(lhs != rhs) );  }
```

Είναι λάθος να γράψουμε "`!(CourseKey(aCode) != cPrereq)`" αφού επιφορτώσαμε ήδη τον "`!=`"; Μια χαρά είναι! Απλώς, αυτό που γράφουμε με τον "`==`" είναι πιο κομψό.

Σημείωση:▶
Πρόσεξε τον τρόπο επιφόρτωσης του "`==`": γίνεται με βάση τον "`!=`". Δεν θα ήταν πιο γρήγορος αν γράφαμε "`return ( strcmp(lhs.s, rhs.s) == 0 )`"; Ναι, αλλά έτσι που το γράψαμε έχουμε σίγουρη συμβατότητα των δύο τελεστών. Αν πάμε για τα πιο γρήγορα μπορεί να κάνουμε κάποιο λάθος, μπορεί, αν χρειαστεί να κάνουμε κάποια αλλαγή, να ξεχάσουμε να την κάνουμε και στις δύο συναρτήσεις και άλλα τέτοια.◀

Η *setTitle()* είναι απλή:

```
void Course::setTitle( string aTitle )
```

[3] Δηλαδή αν θέλουμε να «καθαρίσουμε» ολόκληρο το αντικείμενο δεν θα μπορούμε να «καθαρίσουμε» τον κωδικό; Στην περίπτωση αυτή δεν καθαρίζουμε τα μέλη ένα προς ένα με τις "*set*"· χρησιμοποιούμε τον (ερήμην) δημιουργό χωρίς ορίσματα:

```
   c0 = Course();
```

```
{
   strncpy( cTitle, aTitle.c_str(), cTitleSz-1 );  cTitle[cTitleSz-1] = '\0';
} // Course::setTitle
```

Η

```
void Course::setFSem( int aFSem )
{
   if ( aFSem < 1 || 8 < aFSem)
      throw CourseXptn( cCode, "setFSem", CourseXptn::rangeError, aFSem );
   cFSem = aFSem;
} // Course::setFSem
```

ρίχνει εξαίρεση για την οποίαν θα πρέπει να κάνουμε ορισμένες αλλαγές στην *CourseXptn*:

- Δηλώνουμε άλλο ένα μέλος:

```
   int  errIntVal;
```

- και ορίζουμε άλλον έναν δημιουργό:

```
   CourseXptn( const Course::CourseKey& obk, const char* mn,
               int ec, int iv )
   {  objKey = obk;
      strncpy( funcName, mn, 99 );  funcName[99] = '\0';
      errorCode = ec;
      errIntVal = iv;  }
```

Η *setCompuls()* είναι πολύ απλή:

```
   void setCompuls( bool aCompuls ) { cCompuls = aCompuls; };
```

Να πούμε όμως ότι –για διαχείριση μελών τύπου `bool`– μπορεί να δεις αλλού ένα ζευγάρι μεθόδων:

```
   void setCompuls() {  cCompuls = true;  };
   void clearCompuls() {  cCompuls = false;  };
```

Η αναλλοίωτη μας λέει πώς να γράψουμε τις *setSector()* και *setCateg()*:

```
void Course::setSector( char aSector )
{
   if ( aSector != 'Υ' && aSector != 'Π' &&
        aSector != 'Μ' && aSector != 'Γ' && aSector != 'Ξ' )
      throw CourseXptn( cCode, "setSector",
                        CourseXptn::noSuchSector, aSector );
   cSector = aSector;
} // Course::setSector

void Course::setCateg( string aCateg )
{
   if ( aCateg != "ΜΓΥ" && aCateg != "ΜΕΥ" &&
        aCateg != "ΜΕ" && aCateg != "ΔΟΝ" )
      throw CourseXptn( cCode, "setCateg",
                        CourseXptn::noSuchCateg, aCateg.c_str() );
   strcpy( cCateg, aCateg.c_str() );
} // Course::setSector
```

Οι *setWH()* και *setUnits()* μοιάζουν με την *setFSem()*:

```
void Course::setWH( int aWH )
{
   if ( aWH <= 0 )
      throw CourseXptn( cCode, "setWH", CourseXptn::rangeError, aWH );
   cWH = aWH;
} // Course::setWH

void Course::setUnits( int aUnits )
{
   if ( aUnits <= 0 )
      throw CourseXptn( cCode, "setUnits", CourseXptn::rangeError, aUnits );
   cUnits = aUnits;
} // Course::setUnits
```

και θα έπρεπε να μοιάζουν περισσότερο: θα έπρεπε να υπάρχουν και περιορισμοί εκ των άνω.

Δεν θα γράψουμε *setNoOfStudents()*· ο καλύτερος τρόπος διαχείρισης του μέλους *cNoOfStudents* γίνεται με τις μεθόδους που γράψαμε αρχικώς:

```
   void clearStudents() {  cNoOfStudents = 0;  }
   void add1Student() {  ++cNoOfStudents;  }
```

Τώρα όμως θα πρέπει να προβλέψουμε και περίπτωση διαγραφής φοιτητή από κάποιο μάθημα. Για μια τέτοια περίπτωση θα πρέπει να γράψουμε μια:

```
void Course::delete1Student()
{
   if ( cNoOfStudents <= 0 )
      throw CourseXptn( cCode, "delete1Student", CourseXptn::noStudent );
   --cNoOfStudents;
} // Course::delete1Student
```

Η *setPrereq()* είναι κατ' αρχήν –δηλαδή σύμφωνα με όσα λέει η αναλλοίωτη– απλή:

```
void Course::setPrereq( const string& prCode )
{
   if ( prCode.length() != cCodeSz-1 && prCode.length() != 0 )
      throw CourseXptn( cCode, "setPrereq", CourseXptn::keyLen,
                        prCode.c_str() );
   if ( cCode != CourseKey("") && CourseKey(prCode) == cCode )
      throw CourseXptn( cCode, "setPrereq", CourseXptn::autoRef,
                        prCode.c_str() );
   cPrereq = CourseKey( prCode );
} // Course::setPrereq
```

Χρειάζεται όμως ιδιαίτερη μεταχείριση από το πρόγραμμα που τη χρησιμοποιεί διότι, όπως είπαμε, για να βάλουμε εκεί κάποιον κωδικό μαθήματος *«θα πρέπει επιπλέον να υπάρχει μάθημα με τέτοιον κωδικό στον πίνακα μαθημάτων.»* Με αυτό θα ασχοληθούμε στη συνέχεια.

Τέλος –και εκτός «συνταγής» πια– ας έλθουμε στις *save()* και *load()*. Η *save()* παραμένει σχεδόν όπως ήταν. Οι μοναδικές αλλαγές είναι στις γραμμές που φυλάγουν τα μέλη *cCode* και *cPrereq* που θα γίνουν:

```
// . . .
   bout.write( cCode.s, sizeof(cCode.s) );    // κωδικός μαθήματος
// . . .
   bout.write( cPrereq.s, sizeof(cPrereq.s) );   // προαπαιτούμενο
// . . .
```

Στη *load()* θα κάνουμε μεγαλύτερες αλλαγές: θα εφαρμόσουμε αυτά που είδαμε στην §21.6 για να της δώσουμε *ασφάλεια προς τις εξαιρέσεις* επιπέδου *ισχυρής εγγύησης*. Δηλαδή, αφού *«αποπειράται να αλλάξει την τιμή ενός αντικειμένου ... αν αποτύχει θα πρέπει να αφήνει την παλιά τιμή χωρίς αλλαγές.»*

```
void Course::load( istream& bin )
{
   Course tmp;
   bin.read( tmp.cCode.s, cCodeSz );                        // κωδικός μαθήματος
   if ( !bin.eof() )
   {
      bin.read( tmp.cTitle, cTitleSz );                      // τίτλος μαθήματος
      bin.read( reinterpret_cast<char*>(&tmp.cFSem), sizeof(cFSem) );
                                                             // τυπικό εξάμηνο
      bin.read( reinterpret_cast<char*>(&tmp.cCompuls), sizeof(cCompuls) );
                                                  // υποχρεωτικό ή επιλογής
      bin.read( &tmp.cSector, sizeof(cSector) );                    // τομέας
      bin.read( tmp.cCateg, cCategSz );                           // κατηγορία
      bin.read( reinterpret_cast<char*>(&tmp.cWH), sizeof(cWH) );
                                                        // ώρες ανά εβδομάδα
      bin.read( reinterpret_cast<char*>(&tmp.cUnits), sizeof(cUnits));
                                                       // διδακτικές μονάδες
      bin.read( tmp.cPrereq.s, cCodeSz );                    // προαπαιτούμενο
```

```
      bin.read( reinterpret_cast<char*>(&tmp.cNoOfStudents),
                sizeof(cNoOfStudents) );             // αριθ. φοιτ. στο μάθημα
      if ( bin.fail() )
         throw CourseXptn( cCode, "load", CourseXptn::cannotRead );
      *this = tmp;
   } // if ( !bin.eof() ). . .
} // Course::load
```

Τι κάνουμε εδώ; Δηλώνουμε μια βοηθητική μεταβλητή *tmp* –τύπου *Course*– και αποθηκεύουμε σε αυτήν όσα διαβάζουμε. Αν όλα πάνε καλά αντιγράφουμε στο αντικείμενό μας (`*this`) αυτά που είναι αποθηκευμένα στο *tmp*. Δεν θα χρειαστεί να γράψουμε μια *swap*; Δεν είναι υποχρεωτικό! Εδώ δεν έχουμε δεσμεύσει πόρους του συστήματος, ο τελεστής εκχώρησης δουλεύει σωστά. Έτσι η "`*this = tmp`" κάνει τη δουλειά μας και μάλιστα κάπως ταχύτερα από μια "`swap(tmp)`" (αν τη γράψουμε.)

Η νέα εκδοχή της κλάσης είναι:

```
class Course
{ // version 2
public:
   enum { cCodeSz = 8 };
   struct CourseKey
   {
      char s[cCodeSz];
      explicit CourseKey( string aKey="" )
      {  strncpy( s, aKey.c_str(), cCodeSz-1 );  s[cCodeSz-1] = '\0';  }
   }; // CourseKey
   explicit Course( string aCode="", string aTitle="" );
   ~Course() { };
// getters
   const char* getCode() const {  return cCode.s;  }
   const char* getTitle() const {  return cTitle;  }
   unsigned int getFSem() const {  return cFSem;  }
   bool getCompuls() const {  return cCompuls;  }
   char getSector() const {  return cSector;  }
   const char* geCateg() const {  return cCateg;  }
   unsigned int getWH() const {  return cWH;  }
   unsigned int getUnits() const {  return cUnits;  }
   unsigned int getNoOfStudents() const {  return cNoOfStudents; }
   const char* getPrereq() const {  return cPrereq.s;  }
// setters
   void setCode( string aCode );
   void setTitle( string aTitle );
   void setFSem( int aFSem );
   void setCompuls( bool aCompuls ) {  cCompuls = aCompuls;  };
   void setSector( char aSector );
   void setCateg( string aCateg );
   void setWH( int aWH );
   void setUnits( int aUnits );
   void clearStudents() {  cNoOfStudents = 0;  }
   void add1Student() {  ++cNoOfStudents;  }
   void delete1Student();
   void setPrereq( const string& prCode );
// other
   void save( ostream& bout ) const;
   void load( istream& bin );
private:
   CourseKey    cCode;            // κωδικός μαθήματος
   char         cTitle[cTitleSz]; // τίτλος μαθήματος
   unsigned int cFSem;            // τυπικό εξάμηνο
   bool         cCompuls;         // υποχρεωτικό ή επιλογής
   char         cSector;          // τομέας
   char         cCateg[cCategSz]; // κατηγορία
   unsigned int cWH;              // ώρες ανά εβδομάδα
   unsigned int cUnits;           // διδακτικές μονάδες
   CourseKey    cPrereq;          // προαπαιτούμενο
   unsigned int cNoOfStudents;    // αριθ. φοιτητών
```

```
}; // Course

bool operator!=( const Course::CourseKey& lhs, const Course::CourseKey& rhs )
{  return ( strcmp(lhs.s, rhs.s) != 0 );  }

bool operator==( const Course::CourseKey& lhs, const Course::CourseKey& rhs )
{  return ( !(lhs != rhs) );  }
```

κσι της κλάσης εξαιρέσεων:

```
struct CourseXptn
{ // version 2
   enum { keyLen, rangeError, noSuchSector, noSuchCateg, autoRef, noCourse,
          noStudent, fileNotOpen, cannotWrite, cannotRead };
   Course::CourseKey objKey;
   char              funcName[100];
   int               errorCode;
   char              errStrVal[100];
   int               errIntVal;
   CourseXptn( const Course::CourseKey& obk, const char* mn,
               int ec, const char* sv="" )
      : objKey( obk ),  errorCode( ec )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';
      strncpy( errStrVal, sv, 99 );  errStrVal[99] = '\0';  }
   CourseXptn( const Course::CourseKey& obk, const char* mn,
               int ec, int iv )
      : objKey( obk ),  errorCode( ec ),  errIntVal( iv )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';  }
}; // CourseXptn
```

Prj04.3 ... Και ο «Πίνακας Μαθημάτων»

Στο πρόγραμμα του Project 3 παραστήσαμε τον πίνακα μαθημάτων με έναν απλό δυναμικό πίνακα με στοιχεία τύπου *Course*. Η διαχείρισή του έγινε από το πρόγραμμα εφαρμογής και τις συναρτήσεις του.

Εδώ θα τον διαχειριστούμε πιο «νοικοκυρεμένα»: ως αντικείμενο κλάσης

```
class CourseCollection
{
public:
// . . .
private:
   enum { ccIncr = 30 };
   Course* ccArr;
   size_t  ccNOfCourses;
   size_t  ccReserved;
// . . .
}; // CourseCollection
```

με την οποία θα έχουμε την ευκαιρία να δούμε και μερικά νέα(;) πράγματα.

Η *CourseCollection* είναι μια κλάση για τη διαχείριση *συνόλου* που υλοποιούμε με έναν *δυναμικό πίνακα*.

Σύμφωνα με αυτά που είπαμε, θα χρειαστεί να γράψουμε δημιουργό αντιγραφής και τελεστή εκχώρησης. Εδώ όμως προσοχή: Σε οποιοδήποτε πρόγραμμα (μπορούμε να φανταστούμε) που χρησιμοποιεί αντικείμενα αυτής της κλάσης, καθένα από αυτά θα πρέπει να είναι μοναδικό. Διότι αλλιώς έχουμε το εξής πρόβλημα:

- Πόσα αντίγραφα του προγράμματος σπουδών μπορούμε να έχουμε;
- Με ποιο από αυτά θα κάνουμε τι;

Μια λύση του προβλήματος είναι η *απαγόρευση δημιουργίας αντιγράφων κλάσης CourseCollection*. Πρόσεξε πώς:

```
class CourseCollection
{
public:
```

```
// . . .
private:
   enum { ccIncr = 30 };
   Course* ccArr;
   size_t  ccNOfCourses;
   size_t  ccReserved;

   CourseCollection( const CourseCollection& rhs ) { };
   CourseCollection& operator=( const CourseCollection& rhs ) { };
// . . .
}; // CourseCollection
```

Αν στο πρόγραμμά σου (στη **main**) δώσεις:

```
   CourseCollection allCourses;
// . . .
      CourseCollection otherCollection( allCourses );
```

ο μεταγλωττιστής θα βγάλει σφάλμα:

```
'CourseCollection::CourseCollection(const CourseCollection &)' is not accessi-
ble in function main()
```

ή κάτι παρόμοιο, ενώ αν δώσεις:

```
      CourseCollection otherCollection;
// . . .
      otherCollection = allCourses;
```

θα πάρεις:

```
'CourseCollection::operator =(const CourseCollection &)' is not accessible in
function main()
```

ή κάτι παρόμοιο.

Ας έλθουμε τώρα στην *αναλλοίωτη* που περιέχει –κατ' αρχάς– τη συνθήκη διαχείρισης του δυναμικού πίνακα:

$$I_I \equiv (0 \le ccNOfCourses < ccReserved) \;\&\&\; (ccReserved \;\%\; ccIncr == 0)$$

Αλλά έχουμε και επιπλέον περιορισμούς για τα στοιχεία του πίνακα:

- Με τον πίνακα υλοποιούμε ένα *σύνολο*, ας το πούμε *Collection*. Επομένως, δεν θα πρέπει να υπάρχουν δύο ίδια στοιχεία:

$$\forall x, y: Course \bullet (x, y \in Collection \Rightarrow x \neq y)$$

- Είναι δεκτό ένα στοιχείο του πίνακα με κωδικό μαθήματος μηδενικού μήκους; Όχι! Θα πρέπει λοιπόν να έχουμε:[4]

$$\forall x: Course \bullet (x \in Collection \Rightarrow strlen(x.cCode) == cCodeSz\text{-}1)$$

- Στην §Prj04.2 λέγαμε για το προαπαιτούμενο: «*θα πρέπει επιπλέον να υπάρχει μάθημα με τέτοιον κωδικό στον πίνακα μαθημάτων. Για να μπορέσεις να βάλεις αυτόν τον περιορισμό στην αναλλοίωτη και τον αντίστοιχο έλεγχο στις μεθόδους θα πρέπει να σχεδιάσεις τις κλάσεις σου έτσι ώστε τα αντικείμενα τύπου Course να υπάρχουν μόνον μέσα σε κάποιον "πίνακα μαθημάτων"*». Εδώ λοιπόν θα πρέπει να βάλουμε:[5]

$$\forall x: Course \bullet (x \in Collection \;\&\&\; x.cPrereq \neq \texttt{""})$$
$$\Rightarrow \exists y: Course \bullet (y \in Collection \;\&\&\; y.cCode == x.cPrereq)$$

Η αναλλοίωτη είναι:

$$I \equiv I_D \;\&\&\; I_I$$

όπου το πρώτο κομμάτι είναι:

$$I_D \equiv (\forall x, y: Course \bullet (x, y \in Collection \Rightarrow x \neq y)) \;\&\&$$
$$(\forall x: Course \bullet (x \in Collection \Rightarrow strlen(x.cCode) == cCodeSz\text{-}1)) \;\&\&$$

---

[4] Στις Βάσεις Δεδομένων αυτός ο περιορισμός ονομάζεται **περιορισμός** (constraint) **ακεραιότητας οντότητας** (entity integrity)...

[5] ... και αυτός περιορισμός **ακεραιότητας αναφοράς** (referential integrity).

$$(\forall x: Course \bullet (x \in Collection \;\&\&\; x.cPrereq \neq \texttt{""})$$
$$\Rightarrow \exists y: Course \bullet (y \in Collection \;\&\&\; y.cCode == x.cPrereq))$$

Παρ' όλο που προσπαθήσαμε να γράψουμε την αναλλοίωτη κάπως αφηρημένα είναι σαφές ότι παίρνουμε υπόψη μας ότι το *cCode* είναι κλειδί.

Αν πάρουμε υπόψη μας ότι το σύνολο υλοποιείται με τα πρώτα *ccNOfCourses* στοιχεία του πίνακα ccArr η I_D γίνεται:

$$I_D \equiv (\forall j, k: [0\,..\,ccNOfCourses) \bullet (j \neq k \Rightarrow ccArr[j].cCode \neq ccArr[k].cCode)) \;\&\&$$
$$(\forall k: [0\,..\,ccNOfCourses) \bullet strlen(ccArr[k].cCode) == cCodeSz\text{-}1) \;\&\&$$
$$(\forall k: [0\,..\,ccNOfCourses) \bullet (ccArr[k].cPrereq \neq \texttt{""} \Rightarrow$$
$$\exists j: [0\,..\,ccNOfCourses) \bullet ccArr[j].cCode == ccArr[k].cPrereq))$$

Κατά τα άλλα:

- Έχουμε ερήμην δημιουργό:

```
CourseCollection::CourseCollection()
{
   try
   {
      ccReserved = ccIncr;
      ccArr = new Course[ ccReserved ];
      ccNOfCourses = 0;
   }
   catch( bad_alloc& )
   {
      throw CourseCollectionXptn( "CourseCollection",
                                  CourseCollectionXptn::allocFailed);
   }
} // CourseCollection::CourseCollection
```

- Πρέπει να γράψουμε *καταστροφέα*:

```
   ~CourseCollection()  {  delete[] ccArr;  };
```

- Θα γράψουμε συναρτήσεις "*get*"…

```
   size_t getNOfCourses() const {  return ccNOfCourses;  }
   const Course* getArr() const {  return ccArr;  }
```

- … αλλά δεν θα γράψουμε συναρτήσεις "*set*".

Θα πρέπει να γράψουμε και μεθόδους διαχείρισης των στοιχείων του πίνακα που θα μοιάζουν με αυτές που γράψαμε για τη *Route* (§20.7.2.3). Ξαναδές τι κάναμε με τον πίνακα rAllStops:

- Ο ccArr έχει μια σημαντική ομοιότητα με τον rAllStops: *και οι δύο χρησιμοποιούνται για την παράσταση συνόλων*. Έτσι, όπως είχαμε τις *find1RouteStop()*, *get1RouteStop()*, *delete-RouteStop()* – *erase1RouteStop()* και *addRouteStop()* – *insert1RouteStop()* θα πρέπει να γράψουμε τις *find1Course()*, *get1Course()*, *delete1Course()* – *erase1Course()* και *add1Course()* – *insert1Course()* αντιστοίχως.
- Τα στοιχεία του συνόλου που παριστάνει ο rAllStops έχουν, ανά δύο, διαφορετικό *sName* και διαφορετική *sDist*. Στην περίπτωση του ccArr τα στοιχεία έχουν, ανά δύο, διαφορετικό *cCode*. Ο κωδικός μαθήματος είναι άλλο ένα χαρακτηριστικό παράδειγμα *υποκατάστατου κλειδιού* (§15.5.1), όπως είναι ο αριθμός μητρώου για τη *Student*.
- Στην περίπτωση του rAllStops προτιμήσαμε να επιφορτώσουμε τον "`!=`" με σύγκριση στο μέλος *sDist* διότι είχαμε να χειριστούμε την ταξινόμηση του πίνακα σύμφωνα με την απόσταση από την αφετηρία. Αυτό μας στέρησε τη δυνατότητα χρήσης της *linSearch* –από τη `MyTmpltLib`– για αναζητήσεις στάσεων με βάση το όνομά τους. Εδώ έχουμε να κάνουμε αναζητήσεις με βάση τον κωδικό μαθήματος. Αν θέλουμε να χρησιμοποιήσουμε τη *linSearch()* μπορούμε να επιφορτώσουμε τον "`!=`" για την *Course* με σύγκριση των κωδικών μόνο. Η επιφόρτωση θα μπορούσε να γίνει, όπως στη *Date* (και τη *Student*), με μια καθολική συνάρτηση:

```
bool operator!=( const Course& a, const Course& b )
```

```
{  return ( strcmp(a.getCode(), b.getCode()) != 0 );  }
```

Αυτή περνάει από τον μεταγλωττιστή και το πρόγραμμά μας δουλεύει μια χαρά. Ας πούμε τώρα ότι –για κάποιον λόγο– θέλουμε να αλλάξουμε τη λογική σύγκρισης κλειδιών και αντικειμένων. Θα πρέπει να αλλάξουμε τις συναρτήσεις επιφόρτωσης του "`!=`" της *CourseKey* και της *Course* που υλοποιούν την ίδια λογική. Φυσικά, αυτό εισάγει στο λογισμικό μας μια δυνατότητα για μελλοντικό σφάλμα. Αυτό αποφεύγεται αν ορίσουμε:

```
bool operator!=( const Course& lhs, const Course& rhs )
{  return ( Course::CourseKey(lhs.getCode()) !=
            Course::CourseKey(rhs.getCode()) );  }
```

οπότε η λογική της σύγκρισης υλοποιείται μια φορά μόνον, για τον τύπο *CourseKey*. Πέρα από αυτό, εδώ λέμε ξεκάθαρα ότι δύο αντικείμενα τύπου *Course* είναι διαφορετικά αν και μόνον αν έχουν διαφορετικούς κωδικούς.

- Η –μικρότερης σημασίας– διαφορά του ccArr από τον rAllStops είναι η ταξινόμηση των στοιχείων του rAllStops. Αφού ο ccArr δεν είναι ταξινομημένος δεν χρειάζεται να κάνουμε πολλές μετακινήσεις στοιχείων στη διαγραφή και την εισαγωγή. Στην §20.7.2 δώσαμε ένα σχέδιο για τη *διαγραφή* σε μια τέτοια περίπτωση.

  Αρχίζουμε από την *find1Course()* που θα γίνει:

```
   bool find1Course( const string& code ) const
   {  return ( findNdx(code) >= 0 );  };
```

όπου τώρα:

```
int CourseCollection::findNdx( const string& code ) const
{
   int ndx;
   if ( code.length() != Course::cCodeSz-1 )
      ndx = -1;
   else
      ndx = linSearch( ccArr, ccNOfCourses, 0, ccNOfCourses-1, Course(code) );
   return ndx;
} // CourseCollection::findNdx
```

Πρόσεξε ότι:

- Και εδώ η *findNdx()* επιστρέφει τον δείκτη του στοιχείου (ή "-1" αν η αναζήτηση αποτύχει) ενώ η *find1Course()* είναι υλοποίηση της σχέσης "$\in$" και επιστρέφει `true` ή `false`.
- Και εδώ θα δηλώσουμε τη *findNdx()* στην περιοχή `private` με ίδιο σκεπτικό που είδαμε στην περίπτωση της *Route* (§20.7.2.3).
- Αποκλείεται να κληθεί ο δημιουργός της *Course* αν η τιμή της *code* έχει «παράνομο» μήκος.

  Ας δούμε τώρα την *erase1Course()*. Μετατρέπουμε το σχέδιο που δώσαμε στην §20.7.2:

```
void CourseCollection::delete1Course( string code )
{
   if ( υπάρχει μάθημα με κωδικό code στη θέση ndx )
   {
      αντίγραψε στη θέση ndx το τελευταίο στοιχείο του πίνακα
      --ccNOfCourses;
   }
}
```

και γράφουμε τη μέθοδο:

```
void CourseCollection::delete1Course( string code )
{
   int ndx( findNdx(code) );
   if ( ndx >= 0 )  // code found
   {
      ccArr[ndx] = ccArr[ccNOfCourses-1];
      --ccNOfCourses;
   }
```

```
} // CourseCollection::delete1Course
```

Όπως θα δούμε στη συνέχεια, αυτή η μέθοδος μπορεί να οδηγήσει σε παραβίαση της αναλλοίωτης. Στη συνέχεια θα τη συμπληρώσουμε και θα τη διασπάσουμε σε δύο μεθόδους.

Η *insert1Course()* μικρή σχέση έχει με την *insert1RouteStop()*· είναι πολύ απλούστερη. Αν δεν υπάρχει το στοιχείο που θέλουμε να εισαγάγουμε το αντιγράφουμε στο τέλος του πίνακα. Φυσικά, «μεγαλώνουμε» τον πίνακα αν δεν έχει χώρο για το νέο στοιχείο:

```
void CourseCollection::insert1Course( const Course& aCourse )
{
   if ( ccReserved <= ccNOfCourses+1 )
   {
      try {  renew( ccArr, ccNOfCourses, ccReserved+ccIncr );
             ccReserved += ccIncr;  }
      catch( MyTmpltLibXptn& )
      {
         throw CourseCollectionXptn( "insert1Course",
                                  CourseCollectionXptn::allocFailed );
      }
   }
   ccArr[ccNOfCourses] = aCourse;
   ++ccNOfCourses;
} // CourseCollection::insert1Course
```

Prj04.3.1 Οι Μέθοδοι *add1Course()* και *delete1Course()*

Η δουλειά που κάνει η *insert1Course()* δεν είναι αρκετή για την περίπτωσή μας αφού δεν συμμορφώνεται με την αναλλοίωτη. Έτσι, θα πρέπει να γράψουμε μια άλλη μέθοδο, ας την πούμε *add1Course()*, που θα καλεί την *insert1Course()* αφού κάνει τους εξής ελέγχους:

- Ο κωδικός μαθήματος θα πρέπει να έχει μήκος `cCodeSz-1`:

```
   if ( strlen(aCourse.getCode()) != cCodeSz-1 )
      throw CourseCollectionXptn( "add1Course", CourseCollectionXptn::entity);
```

- Ο πίνακας δεν θα πρέπει να έχει στοιχείο με κωδικό ίδιο με αυτόν του εισαγόμενου:

```
   int ndx( findNdx(aCourse.getCode()) );
   if ( ndx >= 0 )
      throw CourseCollectionXptn( "add1Course", CourseCollectionXptn::key,
                                  aCourse.getCode() );
```

  Αυτό όμως είναι πολύ «αυστηρό»! Αν υπάρχει το στοιχείο δεν το εισάγουμε και τελειώσαμε· η ρίψη εξαίρεσης είναι υπερβολή.

- Αν το εισαγόμενο στοιχείο έχει προαπαιτούμενο αυτό θα πρέπει να υπάρχει ήδη στον πίνακα:

```
   if ( strcmp(aCourse.getPrereq(), "") != 0 )       // υπάρχει προαπαιτούμενο
   {
      int ndx( findNdx(aCourse.getPrereq()) );
      if ( ndx < 0 )              // δεν βρέθηκε το κλειδί του προαπαιτούμενου
         throw CourseCollectionXptn( "add1Course", CourseCollectionXptn::ref,
                                     aCourse.getPrereq() );
   }
```

- Το εισαγόμενο στοιχείο θα πρέπει να μην έχει εγγεγραμμένους φοιτητές. Δηλαδή θα πρέπει να ελέγχουμε το *cNoOfStudents*; Όχι, θα το μηδενίζουμε! Βέβαια, δεν θα πειράζουμε την παράμετρο (περνάει ως "`const Course& aCourse`") αλλά αυτό που εισάχθηκε στον πίνακα:

```
      ndx = findNdx( aCourse.getCode() );
      ccArr[ndx].clearStudents();
```

  Η *add1Course()* θα είναι:

```
void CourseCollection::add1Course( const Course& aCourse )
{
```

```
   if ( strlen(aCourse.getCode()) != Course::cCodeSz-1 )
      throw CourseCollectionXptn( "add1Course",
                                  CourseCollectionXptn::entity );
   int ndx( findNdx(aCourse.getCode()) );
   if ( ndx < 0 ) // δεν βρέθηκε το κλειδί
   {
      if ( strcmp(aCourse.getPrereq(), "") != 0 )// υπάρχει
      {                                          // προαπαιτούμενο
         int ndx( findNdx(aCourse.getPrereq()) );
         if ( ndx < 0 )   // δεν βρέθηκε το κλειδί προαπαιτούμενου
            throw CourseCollectionXptn( "add1Course",
                                        CourseCollectionXptn::ref,
                                        aCourse.getPrereq() );
      }
   // δεν υπάρχει προαπαιτούμενο ή υπάρχει και βρέθηκε στον πίνακα
      insert1Course( aCourse );
      ndx = findNdx( aCourse.getCode() );
      ccArr[ndx].clearStudents();
   }
} // CourseCollection::add1Course
```

Όπως καταλαβαίνεις για να μην έχουμε δυσάρεστες εκπλήξεις θα πρέπει να «κρύψουμε» (**private**) την *insert1Code()*.

Πρόβλημα συμμόρφωσης με την αναλλοίωτη μπορεί να έχουμε και με τη διαγραφή: Αν διαγράψουμε κάποιο στοιχείο που είναι προαπαιτούμενο άλλου (-ων) στοιχείου (-ων). Θα πρέπει λοιπόν να «κρύψουμε» και την *erase1Course()* και να γράψουμε μια *delete1Course()*, που θα καλεί την *erase1Course()* αφού κάνει τους απαραίτητους ελέγχους.

Και κάτι ακόμη: όπως θα θυμάσαι από το Project 3, υπάρχει και ο *Πίνακας Δηλώσεων Μαθημάτων*. Αν το μάθημα που θέλουμε να σβήσουμε έχει *cNoOfStudents* > 0 θα πει ότι

- υπάρχουν στοιχεία του *Πίνακα Δηλώσεων* με τον κωδικό του,
- υπάρχουν αντικείμενα κλάσης *Student* με τον κωδικό του στον πίνακα μαθημάτων στα οποία έχει εγγραφεί ο φοιτητής.

Επομένως δεν μπορούμε να σβήσουμε το μάθημα!

```
void CourseCollection::delete1Course( string code )
{
   int ndx( findNdx(code) );
   if ( ndx >= 0 )  // υπάρχει
   {
      ccArr[ccNOfCourses] = Course();
      ccArr[ccNOfCourses].setPrereq( code ); // φρουρός
      int k(0);
      while ( strcmp(ccArr[k].getPrereq(), code.c_str()) != 0 )
         ++k;
      if ( k < ccNOfCourses )
         throw CourseCollectionXptn( "delete1Course",
                                     CourseCollectionXptn::cannotDel,
                                     code.c_str() );
      int enrStdnt( ccArr[ndx].getNoOfStudents() );
      if ( enrStdnt > 0 ) // υπάρχουν εγγραφές φοιτητών
         throw CourseCollectionXptn( "delete1Course",
                                     CourseCollectionXptn::enrollRef,
                                     code.c_str(), enrStdnt );
      erase1Course( ndx );
   }
} // CourseCollection::delete1Course
```

Αν βρούμε ότι στον πίνακα υπάρχει μάθημα με τον συγκεκριμένο κωδικό (*code*) κάνουμε μια γραμμική αναζήτηση με φρουρό στο μέλος *cPrereq*. Αν βρούμε έστω και ένα στοιχείο του πίνακα που έχει ως προαπαιτούμενο μάθημα με κωδικό *code* ρίχνουμε εξαίρεση.

Ακόμη ρίχνουμε εξαίρεση αν βρούμε `ccArr[ndx].getNoOfStudents() > 0`.

Σημείωση:▶
Αργότερα θα δούμε πώς μπορείς να χειρισθείς με άλλον τρόπο τη διαγραφή μαθήματος στο οποίο υπάρχουν εγγραφές φοιτητών.◀

Αφού οι έλεγχοι πέρασαν στη *delete1Course()*, η *erase1Course()* απλουστεύεται:

```
void CourseCollection::erase1Course( int ndx )
{
   ccArr[ndx] = ccArr[ccNOfCourses-1];
   --ccNOfCourses;
} // CourseCollection::erase1Course
```

Πρόσεξε ότι τώρα, αφού την καλούμε όταν ξέρουμε ότι το μάθημα υπάρχει σίγουρα, την τροφοδοτούμε με τον δείκτη του προς διαγραφή στοιχείου.

Prj04.3.2 Οι *save()* και *load()*

Οι *save()* και *load()* χρησιμοποιούν τις αντίστοιχες της *Course*. Πρώτα η

```
void CourseCollection::save( ofstream& bout )
{
   if ( bout.fail() )
      throw  CourseCollectionXptn( "save", CourseCollectionXptn::fileNotOpen);
   bout.write( reinterpret_cast<const char*>(&ccNOfCourses),
               sizeof(ccNOfCourses) );
   for ( int k(0); k < ccNOfCourses; ++k )
      ccArr[k].save( bout );
   if ( bout.fail() )
      throw  CourseCollectionXptn( "save", CourseCollectionXptn::cannotWrite);
} // CourseCollection::save
```

Η *load()* θέλει πιο πολλή προσοχή, αφού θα πρέπει να έχει *ασφάλεια προς τις εξαιρέσεις επιπέδου ισχυρής εγγύησης*. Δηλαδή, όπως λέγαμε στην §21.6, αφού «*αποπειράται να αλλάξει την τιμή ενός αντικειμένου ... αν αποτύχει θα πρέπει να αφήνει την παλιά τιμή χωρίς αλλαγές.*»

Για να το διασφαλίσουμε θα χρησιμοποιήσουμε ένα τοπικό αντικείμενο

```
   CourseCollection tmp;
```

όπου θα αποθηκεύουμε τις τιμές που διαβάζουμε από το αρχείο. Αν όλα πάνε καλά, θα ανταλλάξουμε την τιμή του *tmp* με την τιμή του ***this**, με χρήση της

```
void CourseCollection::swap( CourseCollection& rhs )
{
   Course* sv( ccArr );
   ccArr = rhs.ccArr;  rhs.ccArr = sv;
   std::swap( ccNOfCourses, rhs.ccNOfCourses );
   std::swap( ccReserved, rhs.ccReserved );
} // CourseCollection::swap
```

Το πρώτο που κάνουμε είναι να διαβάσουμε το πλήθος των στοιχείων του πίνακα σε μια τοπική μεταβλητή:

```
   size_t n;
   bin.read( reinterpret_cast<char*>(&n), sizeof(ccNOfCourses) );
```

Η τιμή της *n* μας δίνει το πλήθος των τιμών τύπου *Course* που ακολουθούν στο αρχείο. Τις διαβάζουμε ως εξής:

```
      for ( int k(0); k < n && !bin.fail(); ++k )
      {
         Course oneCourse;
         oneCourse.load( bin );
         tmp.insert1Course( oneCourse );
      }
```

Πρόσεξε ότι εδώ καλούμε την *insert1Course()* και όχι την *add1Course()* θεωρώντας –ως συνήθως– ότι το περιεχόμενο του μη μορφοποιημένου αρχείου δεν έχει σφάλματα.

Αν η εκτέλεση της **for** τελειώσει επειδή κάτι δεν πήγε καλά ρίχνουμε εξαίρεση. Αλλιώς δίνουμε στο αντικείμενό μας την τιμή του *tmp* με την (ασφαλή) *swap*:

```
      if ( bin.fail() )
         throw  CourseCollectionXptn( "load",
                                      CourseCollectionXptn::cannotRead );
      swap( tmp );
```

Με την ολοκλήρωση εκτέλεσης της *load()* καταστρέφεται το *tmp* και ανακυκλώνεται η δυναμική μνήμη που έχει δεσμεύσει. Το ίδιο θα συμβεί και στην περίπτωση που η εκτέλεση της *load()* διακόπτεται επειδή ρίχνεται κάποια εξαίρεση. Εδώ έχουμε εφαρμογή της τεχνικής *RAII*.

Ολόκληρη η *load*:

```
void CourseCollection::load( ifstream& bin )
{
   CourseCollection tmp;
   unsigned int n;

   bin.read( reinterpret_cast<char*>(&n), sizeof(ccNOfCourses) );
   if ( !bin.eof() )
   {
      for ( int k(0); k < n && !bin.fail(); ++k )
      {
         Course oneCourse;
         oneCourse.load( bin );
         tmp.insert1Course( oneCourse );
      }
      if ( bin.fail() )
         throw  CourseCollectionXptn( "load",
                                      CourseCollectionXptn::cannotRead );
      swap( tmp );
   }
} // CourseCollection::load
```

Prj04.3.3 Απώλειες Πρόσβασης

Κρύβοντας σε περιοχή **private** όλα τα αντικείμενα *Course* αχρηστεύουμε τις ανοιχτές μεθόδους τους. *Αυτά τα αντικείμενα θα παραμείνουν όπως είναι όταν τα βάζουμε για πρώτη φορά*! Δεν μπορούμε να χρησιμοποιήσουμε τη διαδικασία «*get-delete-edit-add*» που είδαμε στη *Route;* Όχι! Θα πάρουμε εξαίρεση

- αν προσπαθήσουμε να σβήσουμε ένα μάθημα που είναι προαπαιτούμενο για κάποιο άλλο·
- αν προσπαθήσουμε να σβήσουμε ένα μάθημα στο οποίο έχουν εγγραφεί φοιτητές (αυτό θα το φροντίσουμε στη συνέχεια).

Ναι, αλλά –πέρα από ανάγκες διόρθωσης λανθασμένων στοιχείων– θα χρειαστεί να αυξάνουμε (μειώνουμε) τον αριθμό των φοιτητών που γράφονται σε (διαγράφονται από) ένα μάθημα. Τι κάνουμε σε αυτήν την περίπτωση; Γράφουμε για την *CourseCollection* ενδιάμεσες μεθόδους που επιτρέπουν την χρήση των μεθόδων των αντικειμένων *Course*. Στην περίπτωσή μας θα δηλώσουμε τις:

```
   void add1Student( string code );
   void delete1Student( string code );
```

και θα τις ορίσουμε:

```
void CourseCollection::add1Student( string code )
{
   int ndx( findNdx(code) );
   if ( ndx < 0 ) // δεν υπάρχει τέτοιο μάθημα
      throw CourseCollectionXptn( "add1Student",
                                  CourseCollectionXptn::notFound,
                                  code.c_str() );
```

```
   ccArr[ndx].add1Student();
} // CourseCollection::add1Student

void CourseCollection::delete1Student( string code )
{
   int ndx( findNdx(code) );
   if ( ndx < 0 ) // δεν υπάρχει τέτοιο μάθημα
      throw CourseCollectionXptn( "delete1Student",
                                  CourseCollectionXptn::notFound,
                                  code.c_str() );
   ccArr[ndx].delete1Student();
} // CourseCollection::delete1Student
```

Τα ονόματα είναι αυτά των αντίστοιχων μεθόδων της *Course*·αν δεν σου αρέσουν άλλαξέ τα.

Παρόμοιες ενδιάμεσες μεθόδους (της *CourseCollection*) μπορείς να γράψεις και για άλλες μεθόδους της *Course*· *μη γράψεις για τη setCode*. Να θυμάσαι ότι γενικώς, από τη στιγμή που το αντικείμενο μπήκε στη συλλογή, *το κλειδί δεν αλλάζει*. Αν αλλάξεις κάποιον κωδικό μαθήματος θα μείνουν «ξεκρέμαστα»

- τα αντικείμενα του πίνακα που έχουν το μάθημα ως προαπαιτούμενο και
- τα αντικείμενα του *Πίνακα Δήλωσης Μαθημάτων* για εγγραφή σε αυτό το μάθημα.

Prj04.3.4 Επιφορτώνουμε τον "[]";

Πιθανότατα θα αναρωτηθείς: «Αφού ένα αντικείμενο της *CourseCollection* είναι ένας πίνακας στοιχείων τύπου *Course* γιατί να μην επιφορτώσουμε τον τελεστή "`[]`";»

Ναι, ένα αντικείμενο της *CourseCollection* είναι ένας πίνακας στοιχείων τύπου *Course* αλλά «κρύβουμε» αυτό το γεγονός για να μην ενθαρρύνουμε την ανάπτυξη εφαρμογών που να στηρίζονται στην παράσταση των δεδομένων του συνόλου. Έτσι, η *get1Course()* υλοποιείται ως εξής:

```
const Course& CourseCollection::get1Course( string code ) const
{
   int ndx( findNdx(code) );
   if ( ndx < 0 )
      throw CourseCollectionXptn( "get1Course",
                                  CourseCollectionXptn::notFound,
                                  code.c_str() );
   return ccArr[ndx];
} // CourseCollection::get1Course
```

Ο δείκτης *ndx* είναι τοπική μεταβλητή και το πρόγραμμα που χρησιμοποιεί αυτήν τη μέθοδο παίρνει αυτό που ζητάει δίνοντας μια τιμή τύπου *string*.

Αν θέλεις μπορείς να επιφορτώσεις τον "`[]`" ως εναλλακτικό τρόπο γραφής της *get1Course*:

```
const Course& CourseCollection::operator[]( string code ) const
{
   int ndx( findNdx(code) );
   if ( ndx < 0 )
      throw CourseCollectionXptn( "get1Course",
                                  CourseCollectionXptn::notFound,
                                  code.c_str() );
   return ccArr[ndx];
} // CourseCollection::operator[]
```

Αλλά ποιο από τα παρακάτω είναι προτιμότερο:

```
   get1C = allCourses.get1Course( "EY02010" );
   get1C = allCourses["EY02010"];
```

Το δεύτερο φαίνεται λίγο «παράξενο»: μεταξύ των "`[`" και "`]`" έχουμε συνηθίσει να βλέπουμε παράσταση που δίνει έναν φυσικό αριθμό, τον δείκτη για το στοιχείο του πίνακα.

Υπάρχει και ένα άλλο πρόβλημα: Συνήθως ο τύπος του αποτελέσματος της συνάρτησης επιφόρτωσης του "`[]`" είναι *τύπος αναφοράς χωρίς το* "`const`". Έτσι μπορούμε να αλλάζουμε την τιμή του στοιχείου που μας δίνει. Εδώ δεν θέλουμε να δώσουμε τέτοια δυνατότητα.

Όπως βλέπεις, μια τέτοια επιφορτωση θα ήταν ανορθόδοξη από πολλές απόψεις και θα προτιμήσουμε να την αποφύγουμε. Αργότερα θα δούμε ότι κάτι τέτοιο γίνεται στην STL.

Prj04.4 Περί Διαγραφών

Είδαμε πιο πάνω την *CourseCollection*::*delete1Course()* και μην αμφιβάλλεις για τη συνέχια: θα δούμε και *StudentCollection*::*delete1Student()* και *StudentInCourseCollection*::*delete1StudentInCourse()*. Κάνουμε βεβαίως ελέγχους, αλλά σβήνουμε μάλλον εύκολα.

Σκέψου όμως τα εξής: τα διαγραφόμενα μπορεί να περιέχουν στοιχεία που έχουν εισαχθεί με το χέρι· αυτή είναι μια «ακριβή» διαδικασία. Τέτοια στοιχεία δεν τα σβήνουμε «τελεσιδίκως» διότι είναι πιθανό να χρειαστεί να εισαχθούν εκ νέου στους πίνακές μας (όπως είναι ή μετά από κάποια διόρθωση). Να θυμάσαι ότι γενικώς:

- ***Σπανίως σβήνουμε δεδομένα που έχουν εισαχθεί σε μια ΒΔ με το χέρι ή άλλη χρονοβόρα –και επομένως «ακριβή»– διαδικασία.***

Αυτό που μπορείς να κάνεις είναι να γράψεις τις μεθόδους διαγραφής έτσι ώστε τα αντικείμενα που θα τα αφαιρέσει από κάποιον πίνακα να μεταφέρονται σε έναν άλλον πίνακα με την ίδια (περίπου) δομή. Τι εννοούμε με το «περίπου»; Συνήθως κάθε αντικείμενο που εισάγεται στα «διαγραφέντα» συνοδεύεται από τον χρόνο της διαγραφής.

Prj04.5 Η Κλάση *Student*

Η *Student* θα είναι τώρα:

```
class Student
{
public:
// . . .
private:
   enum { sNameSz = 20 };
   enum { sIncr = 3 };
   unsigned int       sIdNum;                  // αριθμός μητρώου
   char               sSurname[sNameSz];
   char               sFirstname[sNameSz];
   unsigned int       sWH;                     // ώρες ανά εβδομάδα
   size_t             sNoOfCourses;            // αριθμός μαθημάτων που δήλωσε
   Course::CourseKey* sCourses;
   size_t             sReserved;
}; // Student
```

Το βέλος *sCourses* θα δείχνει τον δυναμικό πίνακα όπου «*θα αποθηκεύονται ... οι κωδικοί μαθημάτων στα οποία έχει εγγραφεί ο φοιτητής.*»

Η –αντίστοιχη της *Student*– κλάση εξαιρέσεων, όπως –σχεδόν– την έχουμε από το Project 3 είναι:

```
struct StudentXptn
{
   enum { incomplete, negIdNum, nonPosIncr,
          fileNotOpen, cannotRead, cannotWrite };
   unsigned int objKey;
   char         funcName[100];
   int          errorCode;
   int          errIntVal;
   StudentXptn( int obk, const char* mn, int ec, int ev = 0 )
```

```
      :  objKey( obk ),  errorCode( ec ),  errIntVal( ev )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';  }
}; // StudentXptn
```

Η μόνη διαφορά –προς το παρόν– είναι η εισαγωγή ενός ακόμη μελους, του *objKey* και η αλλαγή στον δημιουργό ώστε να το χειριστεί.

Τώρα, ας εφαρμόσουμε τη συνταγή μας:

- **ΑΝ**: Ας ξεκινήσουμε με την αναλλοίωτη της κλάσης:

$$0 \texttt{ <= } sIdNum \texttt{ \&\& } strlen(sSurname) < sNameSz \texttt{ \&\& } strlen(sFirstname) < sNameSz \texttt{ \&\& } sWH \texttt{ >= } 0 \texttt{ \&\&}$$
$$(\forall k: [0..sNoOfCourses) \bullet (strlen(\text{sCourses}[k]) \texttt{ == } Course\texttt{::}cCodeSz\text{-}1)) \texttt{ \&\&}$$
$$(\forall j, k: [0 .. sNoOfCourses) \bullet (j \neq k \Rightarrow \text{sCourses}[j] \neq \text{sCourses}[k])) \texttt{ \&\&}$$
$$(0 \leq sNoOfCourses < sReserved) \texttt{ \&\& } (sReserved \texttt{ \% } sIncr \texttt{ == } 0)$$

Να παρατηρήσουμε τα εξής:

1) Το *sIdNum* μπορεί να πάρει τιμή "`0`" μόνον από τον ερήμην δημιουργό σε δήλωση της μορφής:

```
   Student s1;
```

2) Για το μέλος *sIdNum* χρειαζόμαστε πιο «περιοριστική» συνθήκη· με τη "0 <= *sIdNum*" γίνεται δεκτός ο αριθμός μητρώου "`1000000`" που είναι μάλλον απίθανο να υπάρχει για κάποιο τμήμα.

3) Όπως στην περίπτωση του *cTitle* της *Course* έτσι και για τα *sSurname* και *sFirstname*, δεν θα ρίχνουμε εξαίρεση αν μας έλθει τιμή με μήκος *sNameSz* ή μεγαλύτερο. Απλώς θα αποθηκεύουμε τους πρώτους *sNameSz*-1 χαρακτήρες.

- **ΔΕ**: Θα χρειαστούμε φυσικά έναν ερήμην δημιουργό.

```
Student:: Student( int aIdNum )
{
   if ( aIdNum < 0 )
      throw StudentXptn( 0, "Student", StudentXptn::negIdNum, aIdNum );
   sIdNum = aIdNum;
   sSurname[0] = '\0';
   sFirstname[0] = '\0';
   sWH = 0;
   try  {  sCourses = new Course::CourseKey[ sIncr ];  }
   catch( bad_alloc )
   {  throw StudentXptn( sIdNum, "Student", StudentXptn::allocFailed );  }
   sReserved = sIncr;
   sNoOfCourses = 0;
}; // Student()
```

Πρόσεξε τη διαφορά των δύο εξαιρέσεων:

- Η πρώτη ρίχνεται αν η *aIdNum* έχει παράνομη τιμή οπότε και δεν μπορούμε να βάλουμε τιμή στο *sIdNum* που είναι κλειδί.
- Στη δεύτερη περίπτωση το *sIdNum* έχει τιμή (που πήρε από την *aIdNum*.)

Prj04.5.1 Ο «Κανόνας των Τριών»

- **ΔΑ**: Θα χρειαστούμε και έναν δημιουργό αντιγραφής; Ο πίνακας μαθημάτων θα είναι δυναμικός. Έτσι, τα αντικείμενα της κλάσης μας δεσμεύουν πόρους του συστήματος (μνήμη) και θα χρειαστεί να γράψουμε δημιουργό αντιγραφής αφού αυτός που μας δίνει αυτομάτως ο μεταγλωττιστής δεν είναι σωστός.
- **ΤΕ**, **ΚΑ**: Για τον ίδιο λόγο (δυναμικός πίνακας) θα χρειαστεί να γράψουμε τελεστή εκχώρησης και καταστροφέα.

  Ο καταστροφέας είναι απλός:

```
   ~Student() {  delete[] sCourses;  };
```

  Και ο δημιουργός αντιγραφής:

```
Student::Student( const Student& rhs )
```

```
{
   sIdNum = rhs.sIdNum;
   strcpy( sSurname, rhs.sSurname );
   strcpy( sFirstname, rhs.sFirstname );
   sWH = rhs.sWH;
   try  {  sCourses = new Course::CourseKey[ rhs.sReserved ];  }
   catch( bad_alloc )
   {  throw StudentXptn( sIdNum, "Student", StudentXptn::allocFailed );  }
   sReserved = rhs.sReserved;
   for ( int k(0); k < rhs.sNoOfCourses; ++k )
      sCourses[k] = rhs.sCourses[k];
   sNoOfCourses = rhs.sNoOfCourses;
}; // Student( const Student& rhs )
```

Και ο (αντιγραφικός) τελεστής εκχώρησης:

```
Student& Student::operator=( const Student& rhs )
{
   if ( &rhs != this )
   {
      try {  Student tmp( rhs );
             swap( tmp );  }
      catch( StudentXptn& x )
      {  strcpy( x.funcName, "operator=" );
         throw;  }
   }
   return *this;
}; // Student( const Student& rhs )
```

όπου

```
void Student::swap( Student& rhs )
{
   std::swap( sIdNum, rhs.sIdNum );

   char svs[sNameSz];
   strcpy( svs, sSurname );  strcpy( sSurname, rhs.sSurname );
   strcpy( rhs.sSurname, svs );

   strcpy( svs, sFirstname );
   strcpy( sFirstname, rhs.sFirstname );
   strcpy( rhs.sFirstname, svs );

   std::swap( sWH, rhs.sWH );
   std::swap( sNoOfCourses, rhs.sNoOfCourses );

   Course::CourseKey* svck( sCourses );
   sCourses = rhs.sCourses;  rhs.sCourses = svck;

   std::swap( sReserved, rhs.sReserved );
} // Student::swap
```

Prj04.5.2 Μέθοδοι "*get*" και "*set*"

GE: Θα χρειαστούμε μεθόδους *"get"* για όλα τα μέλη εκτός από το *sReserved* («μυστικό» της υλοποίησης):

```
   unsigned int getIdNum() const { return sIdNum; }
   const char* getSurname() const { return sSurname; }
   const char* getFirstname() const { return sFirstname; }
   unsigned int getWH() const {  return sWH;  }
   unsigned int getNoOfCourses() const { return sNoOfCourses; }
   const Course::CourseKey* getCourses() const { return sCourses;}
```

SE: Θα χρειαστούμε μεθόδους *"set"* για όλα τα μέλη –κατ' αρχήν– εκτός από το *sReserved*.

Πρώτα η

```
void Student::setIdNum( int aIdNum )
{
```

```
   if ( aIdNum <= 0 )
      throw StudentXptn( sIdNum, "setIdNum", StudentXptn::negIdNum, aIdNum );
   sIdNum = aIdNum;
} // Student::setIdNum
```

Πρόσεξε ότι εδώ δεν επιτρέπουμε τιμή "0", που επιτρέπεται (μόνο) στον ερήμην δημιουργό.

Άλλες δύο, οι *setSurname()* και *setFirstname()*, είναι απλές:

```
void Student::setSurname( string aSurname )
{
   strncpy( sSurname, aSurname.c_str(), sNameSz-1 );
   sSurname[sNameSz-1] = '\0';
} // Student::setSurname

void Student::setFirstname( string aFirstname )
{
   strncpy( sFirstname, aFirstname.c_str(), sNameSz-1 );
   sFirstname[sNameSz-1] = '\0';
} // Student::setSurname
```

Στην περίπτωση (μάλλον απίθανη) που θα έλθει κάποιο επώνυμο ή όνομα με περισσότερους από 19 χαρακτήρες θα αντιγραφούν μόνον οι πρώτοι 19 και φυσικά δεν θα ρίξουμε εξαίρεση. Πάντως, καλό θα είναι να μάθει ο χρήστης του προγράμματος-πελάτη, με κάποιον τρόπο, ότι μια μέθοδος της κλάσης αλλοίωσε τα δεδομένα του!

Θα μπορούσαμε να γράψουμε και *μια* μέθοδο (*setCourses*) που θα δίνει τιμή στο *sNoOfCourses* και τον πίνακα sCourses. Δεν θα το κάνουμε· αν θέλεις κάνε το σαν άσκηση. Πάντως, μια τέτοια μέθοδος δεν θα ήταν και τόσο χρήσιμη αφού στη συνέχεια θα γράψουμε τις *insert1Course()* και *add1Course()*.

Στις μεθόδους "*set*" θα περιλάβουμε και την

```
   void clearCourses() {  sNoOfCourses = 0;  sWH = 0;  }
```

που είχαμε στο Project 3. Αυτή παραμένει όπως ήταν! Και ο πίνακας; Ο πίνακας παραμένει όπως είναι, η τιμή του *sReserved* δεν αλλάζει, αλλά η τιμή του *sNoOfCourses* (== 0) μας λέει ότι δεν υπάρχουν στοιχεία για επεξεργασία. Θα μπορούσαμε βεβαίως να επιστρέψουμε τη μνήμη του δυναμικού πίνακα και να πάρουμε νέα με *sIncr* στοιχεία μόνον. Δεν αξίζει τον κόπο...

Prj04.5.3 Μέθοδοι για τα Στοιχεία του Πίνακα

Κατ' αρχήν μπορούμε να χρησιμοποιήσουμε ως οδηγό τις αντίστοιχες μεθόδους της *CourseCollection* –και αυτό θα κάνουμε– εστιάζοντας την προσοχή μας στις διαφορές.

Η βασική μέθοδος θα είναι η *findNdx()* που τροφοδοτείται με έναν κωδικό μαθήματος και μας επιστρέφει τη θέση του στον πίνακα ή, αν δεν υπάρχει, "–1". Την αντιγράφουμε (σχεδόν) από την *CourseCollection*:

```
int Student::findNdx( const string& code ) const
{
   int ndx;
   if ( code.length() != Course::cCodeSz-1 )
      ndx = -1;
   else
      ndx = linSearch( sCourses, sNoOfCourses, 0, sNoOfCourses-1,
                       Course::CourseKey(code) );
   return ndx;
} // Student::findNdx
```

Η *find1Course()* θα είναι:

```
   bool find1Course( const string& code ) const
   {  return ( findNdx(code) >= 0 );  };
```

Φυσικά, κάθε πρόγραμμα-πελάτης μπορεί να χρησιμοποιεί μόνον τη δεύτερη ενώ η πρώτη θα είναι «κρυμμένη» στην περιοχή `private`.

Θα γράψουμε *get1Course()*; Δεν απαγορεύεται αλλά δεν έχει και νόημα, αφού θα είναι μια συνάρτηση που θα τροφοδοτείται με έναν κωδικό και θα επιστρέφει τον ίδιο κωδικό.

Θα υλοποιήσουμε τώρα την *insert1Course()*:

```
void Student::insert1Course( const Course::CourseKey& aCode )
{
   if ( sReserved <= sNoOfCourses+1 )
   {
      try {  renew( sCourses, sNoOfCourses, sReserved+sIncr );
             ccReserved += ccIncr;  }
      catch( MyTmpltLibXptn& )
      {
         throw StudentXptn( sIdNum, "insert1Course",
                                    StudentXptn::allocFailed );
      }
   }
   sCourses[sNoOfCourses] = aCode;
   ++sNoOfCourses;
} // Student::insert1Course
```

Πρόσεξε όμως ότι η μέθοδος αυτή

- Δεν εξασφαλίζει τη μοναδικότητα των κωδικών στον πίνακα.
- Δεν ενημερώνει το μέλος *sWH*.
- Δεν ελέγχει αν υπάρχει στον πίνακα μαθημάτων μάθημα με τέτοιον κωδικό (*ακεραιότητα αναφοράς*).

Για να αποτρέψουμε απρόσεκτη –και εν δυνάμει καταστροφική– χρήση της μεθόδου θα την κρύψουμε, όπως κάναμε και με τη *insert1Course()* της *CourseCollection*· για εξωτερική χρήση θα ξαναγράψουμε την *add1Course()* έτσι που να αντιμετωπίζει τα πρώτα δύο προβλήματα. Αργότερα θα αντιμετωπίσουμε και το τρίτο πρόβλημα.

```
void Student::add1Course( const Course& oneCourse )
{
   if ( findNdx(oneCourse.getCode()) < 0 )
   {
      insert1Course( Course::CourseKey(oneCourse.getCode()) );
      sWH += oneCourse.getWH();
   }
} // Student::add1Course
```

Πρόσεξε ότι, παρ' όλο που στην *insert1Course()* θα περάσουμε μόνον τον κωδικό, τροφοδοτούμε τη μέθοδο με ολόκληρο το αντικείμενο κλάσης *Course*· και αυτό διότι στην επόμενη γραμμή παίρνουμε από αυτό και τις ώρες εβδομαδιαίας διδασκαλίας (*getWH()*).

Με παρόμοια λογική γράφουμε και τις:

```
void Student::delete1Course( const Course& oneCourse )
{
   int ndx( findNdx(oneCourse.getCode()) );
   if ( ndx >= 0 )  // υπάρχει
   {
      erase1Course( ndx );
      sWH -= oneCourse.getWH();
   }
} // Student::delete1Course

void Student::erase1Course( int ndx )
{
   sCourses[ndx] = sCourses[sNoOfCourses-1];
   --sNoOfCourses;
} // Student::erase1Course
```

Prj04.5.4 Φύλαξη και Φόρτωση

Για τη φύλαξη ενός αντικειμένου *Student* γράφουμε τη μέθοδο:

```
void Student::save( ostream& bout ) const
{
   if ( bout.fail() )
      throw StudentXptn( sIdNum, "save", StudentXptn::fileNotOpen );
   bout.write( reinterpret_cast<const char*>(&sIdNum), sizeof(sIdNum) );
   bout.write( sSurname, sizeof(sSurname) );
   bout.write( sFirstname, sizeof(sFirstname) );
   bout.write( reinterpret_cast<const char*>(&sWH), sizeof(sWH) );
   bout.write( reinterpret_cast<const char*>(&sNoOfCourses),
               sizeof(sNoOfCourses) );
   for ( int k(0); k < sNoOfCourses; ++k )
      bout.write( sCourses[k].s, Course::cCodeSz );
   if ( bout.fail() )
      throw StudentXptn( sIdNum, "save", StudentXptn::cannotWrite );
} // Student::save
```

που διαφέρει από αυτήν της αρχικής εκδοχής στο ότι εδώ υπάρχει η **for** που φυλάγει τους κωδικούς των μαθημάτων.

Η *load()* διαφέρει πιο πολύ από την αρχική.

```
void Student::load( istream& bin )
{
   Student tmp;
   bin.read( reinterpret_cast<char*>(&tmp.sIdNum), sizeof(sIdNum) );
   if ( !bin.eof() )
   {
      bin.read( tmp.sSurname, sizeof(sSurname) );
      bin.read( tmp.sFirstname, sizeof(sFirstname) );
      bin.read( reinterpret_cast<char*>(&tmp.sWH), sizeof(sWH) );
      bin.read( reinterpret_cast<char*>(&tmp.sNoOfCourses),
                sizeof(sNoOfCourses) );
      if ( tmp.sNoOfCourses >= tmp.sReserved )
      {
         delete[] tmp.sCourses;
         try
         {
            tmp.sCourses =
                    new Course::CourseKey[((tmp.sNoOfCourses/sIncr)+1)*sIncr];
            tmp.sReserved = ((tmp.sNoOfCourses/sIncr)+1)*sIncr;
         }
         catch( bad_alloc )
         {
            throw StudentXptn( tmp.sIdNum, "load", StudentXptn::allocFailed );
         }
      }
      for ( int k(0); k < tmp.sNoOfCourses; ++k )
         bin.read( tmp.sCourses[k].s, Course::cCodeSz );
      if ( bin.fail() )
         throw StudentXptn( sIdNum, "load", StudentXptn::cannotRead );
      swap( tmp );
   }
} // Student::load
```

Εδώ κάνουμε τη φόρτωση με κάπως διαφορετικό τρόπο από αυτόν που χρησιμοποιήσαμε στην *CourseCollection*. Και στις δύο περιπτώσεις αποθηκεύουμε αυτά που διαβάζουμε σε ένα τοπικό αντικείμενο (*tmp*) και τελικώς ανταλλάσσουμε την τιμή του με το `*this`:

- Εκεί, διαβάζαμε τα αντικείμενα τύπου *Course* και τα βάζαμε στον πίνακα με την *insert1Course()*. Η μέθοδος αυτή μετράει τα στοιχεία του πίνακα (αυξάνει τον *sNoOfCourses*). Το πλήθος των στοιχείων που διαβάστηκε από το αρχείο αποθηκεύεται σε μια τοπική μεταβλητή *n*. Ακόμη, η *insert1Course()*, παίρνει και τη δυναμική μνήμη όταν χρειαστεί.
- Εδώ, το πλήθος των στοιχείων που διαβάζεται από το αρχείο αποθηκεύεται στο μέλος του αντικειμένου *tmp*.*sNoOfCourses*. Αν χρειάζεται, παίρνουμε την απαραίτητη δυναμι-

κή μνήμη, αφού πρώτα ανακυκλώσουμε όση μνήμη είχε ήδη δεσμεύσει το αντικείμενο. Στη συνέχεια αυτά που διαβάζουμε απόοθηκεύονται κατ' αυθείαν στα στοιχεία του πίνακα.

Εκτός από τις *save()* και *load()*, στην πρώτη εκδοχή της κλάσης, είχαμε και μια άλλη μέθοδο για ανάγνωση από αρχείο, τη *readFromText()*. Θα την κρατήσουμε και εδώ αλλά θα τις δώσουμε πιο «ρεαλιστικό» όνομα: *readPartFromText()*. Θα της κάνουμε όμως και μια άλλη αλλαγή: δεν θα αλλάζει την τιμή του αντικειμένου σε περίπτωση που θα ρίξει εξαίρεση:

```
void Student::readPartFromText( istream& tin )
{
   string line;
   getline( tin, line, '\n' );
   if ( !tin.eof() )
   {
      Student tmp;
      size_t t1Pos( line.find("\t") );
      if ( t1Pos >= line.length() )
         throw StudentXptn( sIdNum, "readPartFromText",
                            StudentXptn::incomplete );
      string str1( line.substr(0, t1Pos) );
      int    iStr1( atoi(str1.c_str()) );
'     try {  tmp.setIdNum( iStr1 );  }
      catch( StudentXptn& x )
      {  strcpy( x.funcName, "readPartFromText" );
         throw;  }
      size_t t2Pos( line.find("\t", t1Pos+1) );
      if ( t2Pos >= line.length() )
         throw StudentXptn( tmp.sIdNum, "readPartFromText",
                            StudentXptn::incomplete );
      tmp.setSurname( line.substr(t1Pos+1, t2Pos-t1Pos-1) );
      tmp.setFirstname( line.substr(t2Pos+1) );
      swap( tmp );
   } // if ( !tin.eof . . .
} // Student::readPartFromText
```

Prj04.5.5 Η Κλάση *Student*

Να πώς έγινε τώρα η κλάση *Student*:

```
class Student
{ // version 2
public:
// constructors, destructor
   explicit Student( int aIdNum=0 );
   Student( const Student& rhs );
   ~Student() {  delete[] sCourses;  };
// copy assignement
   Student& operator=( const Student& rhs );
// getters
   unsigned int getIdNum() const { return sIdNum; }
   const char* getSurname() const { return sSurname; }
   const char* getFirstname() const { return sFirstname; }
   unsigned int getWH() const {  return sWH;  }
   unsigned int getNoOfCourses() const { return sNoOfCourses; }
   const Course::CourseKey* getCourses() const
                                          {  return sCourses;  }
// setters
   void setIdNum( int aIdNum );
   void setSurname( string aSurname );
   void setFirstname( string aFirstname );
   void clearCourses() {  sNoOfCourses = 0;  sWH = 0;  }
// 1 Course methods
   bool find1Course( const string& code ) const
   {  return ( findNdx(code) >= 0 );  };
   void add1Course( const Course& oneCourse );
```

```
   void delete1Course( const Course& oneCourse );
// other methods
   void swap( Student& rhs );
   void readPartFromText( istream& tin );
   void save( ostream& bout ) const;
   void load( istream& bin );
private:
   enum { sNameSz = 20 };
   enum { sIncr = 3 };
   unsigned int       sIdNum;        // αριθμός μητρώου
   char               sSurname[sNameSz];
   char               sFirstname[sNameSz];
   unsigned int       sWH;           // ώρες ανά εβδομάδα
   unsigned int       sNoOfCourses;  // αριθμός μαθημάτων που
                                     // δήλωσε
   Course::CourseKey* sCourses;
   unsigned int       sReserved;

   unsigned int countTabs( string aLine );
   int findNdx( const string& code ) const;
   void insert1Course( const Course::CourseKey& aCode );
   void erase1Course( int ndx );
}; // Student
```

Από την προηγούμενη εκδοχή έχουμε και τις

```
bool operator!=( const Student& lhs, const Student& rhs )
{  return (lhs.getIdNum() != rhs.getIdNum());  }

bool operator==( const Student& lhs, const Student& rhs )
{  return !(lhs != rhs);  }
```

Η κλάση εξαιρέσεων είναι:

```
struct StudentXptn
{ // version 2
   enum { allocFailed, outOfRange, incomplete, negIdNum, nonPosIncr,
          fileNotOpen, cannotRead, cannotWrite };
   unsigned int objKey;
   char         funcName[100];
   int          errorCode;
   char         errStrVal[100];
   int          errIntVal;
   StudentXptn( int obk, const char* mn, int ec, const char* sv="" )
      :  objKey( obk ),  errorCode( ec )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';
      strncpy( errStrVal, sv, 99 );  errStrVal[99] = '\0';  }
   StudentXptn( int obk, const char* mn, int ec, int ev )
      :  objKey( obk ),  errorCode( ec ),  errIntVal( ev )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';  }
}; // StudentXptn
```

Prj04.6 Το «Μητρώο Φοιτητών»

Παίρνουμε τα αρχεία `CourseCollection.h` και `CourseCollection.cpp` και με τη *"find-and-replace"* του κειμενογράφου αντικαθιστούμε το "`Course`" με "`Student`". Αυτό που παίρνουμε είναι –σχεδόν– η κλάση *StudentCollection*. Κάθε αντικείμενό της είναι ένα σύνολο αντικειμενων *Student* υλοποιημένο με δυναμικό πίνακα. Μετά από μερικές διορθώσεις έχουμε:

```
class StudentCollection // version 1
{
public:
   StudentCollection();
   ~StudentCollection()  {  delete[] scArr;  };
// getters
   size_t getNOfStudents() const {  return scNOfStudents;  }
   const Student* getArr() const {  return scArr;  }
```

```
// 1 Student
   bool find1Student( int aIdNum ) const
   {  return ( findNdx(aIdNum) >= 0 );  };
   void delete1Student( int aIdNum );
   void add1Student( const Student& aStudent );
   const Student& get1Student( int aIdNum ) const;
// other
   void save( ofstream& bout ) const;
   void load( ifstream& bin );
   void swap( StudentCollection& rhs );
private:
   enum { scIncr = 30 };
   Student* scArr;
   size_t   scNOfStudents;
   size_t   scReserved;

   StudentCollection( const StudentCollection& rhs ) { };
   StudentCollection& operator=( const StudentCollection& rhs ){};
   void erase1Student( int ndx );
   void insert1Student( const Student& aStudent );
   int findNdx( int aIdNum ) const;
}; // StudentCollection
```

Όπως στην *CourseCollection* έτσι και εδώ, προσπαθούμε να δυσκολέψουμε τη δημιουργία αντιγράφων (πόσα μητρώα φοιτητών θα υπάρχουν;) αχρηστεύοντας και «κρύβοντας» δημιουργό αντιγραφής και τελεστή εκχώρησης.

Θα εστιάσουμε την προσοχή μας στις δύο μεθόδους που έχουν διαφορές από τις αντίστοιχες της *CourseCollection*: τη *delete1Student()* και την *add1Student()*. Η πρώτη είναι απλή:

```
void StudentCollection::delete1Student( int aIdNum )
{
   int ndx( findNdx(aIdNum) );
   if ( ndx >= 0 )  // υπάρχει
   {
      if ( scArr[ndx].getNoOfCourses() > 0 )
         throw StudentCollectionXptn( "delete1Student",
                                      StudentCollectionXptn::enrollRef,
                                      aIdNum );
      erase1Student( ndx );
   }
} // StudentCollection::delete1Student
```

αφού αρνούμαστε να συζητήσουμε διαγραφή στην περίπτωση που ο φοιτητής είναι γραμμένος σε μαθήματα· όπως στην *CourseCollection*::*delete1Course()*, αρνούμαστε να διαγράψουμε ένα μάθημα στο οποίο είναι γραμμένοι φοιτητές. Αργότερα θα δούμε πώς μπορείς να γράψεις μια πληρέστερη μέθοδο.

Η δεύτερη έχει κληρονομήσει ένα πρόβλημα από τη *Student*· το πρόβλημα της *insert1Course()* που δεν αντιμετώπισε ούτε η *add1Course()*: «*Δεν ελέγχει αν υπάρχει στον πίνακα μαθημάτων μάθημα με τέτοιον κωδικό (ακεραιότητα αναφοράς).*» Τώρα όμως θα πρέπει να το λύσουμε: δεν θα πρέπει να εισάγουμε στο «μητρώο φοιτητών» αντικείμενα αν δεν σιγουρεύουμε ότι όλοι οι κωδικοί μαθημάτων αντιστοιχούν σε μαθήματα του πίνακα μαθημάτων (αν μας δίδεται).

Τι θα κάνουμε; Να βάλουμε τον πίνακα μαθημάτων μέσα στη *StudentCollection*; Όχι! Ο πίνακας μαθημάτων είναι αυθύπαρκτος και χρήσιμος και για άλλες δουλειές. Μια σαφώς καλύτερη λύση είναι να βάλουμε ένα βέλος προς τον πίνακα μαθημάτων:

```
   CourseCollection* scPAllCourses;
```

που του δίνουμε αρχική τιμή στον δημιουργό:

```
StudentCollection::StudentCollection()
{
   try
   {
      scReserved = scIncr;
```

```
      scArr = new Student[ scReserved ];
      scNOfStudents = 0;
   }
   catch( bad_alloc& )
   {
      throw StudentCollectionXptn( "StudentCollection",
                                   StudentCollectionXptn::allocFailed);
   }
   scPAllCourses = 0;
} // StudentCollection::StudentCollection
```

Για να μπορούμε να χειριστούμε την τιμή του βέλους γράφουμε μεθόδους:

```
CourseCollection* getPAllCourses() const {  return scPAllCourses;  }
void setPAllCourses( CourseCollection* pCourses )
{  scPAllCourses = pCourses;  }
```

Μετά απο'αυτά, αν στο πρόγραμμά μας έχουμε δηλώσει:

```
CourseCollection allCourses;
StudentCollection allStudents;
```

μπορούμε να δώσουμε:

```
allStudents.setPAllCourses( &allCourses );
```

Ας γράψουμε τώρα την *add1Student()*:

```
void StudentCollection::add1Student( const Student& aStudent )
{
   int ndx( findNdx(aStudent.getIdNum()) );
   if ( ndx < 0 ) // δεν βρέθηκε το κλειδί
   {
      if ( scPAllCourses != 0 )
      {
         const Course::CourseKey* aStCourses( aStudent.getCourses() );
         for ( int k(0); k < aStudent.getNoOfCourses(); ++k )
         {
           if ( !scPAllCourses->find1Course(aStCourses[k].s) )
              throw StudentCollectionXptn( "add1Student",
                                           StudentCollectionXptn::ref,
                                           aStCourses[k].s );
         }
      }
      insert1Student( aStudent );
   }
} // StudentCollection::add1Student
```

Τα βασικά σημεία:

- Ο έλεγχος για την μοναδικότητα του κλειδιού (αντικειμένου) γίνεται οπωσδήποτε (`findNdx(aStudent.getIdNum()) < 0`).
- Ο έλεγχος των κωδικών των μαθημάτων γίνεται μόνο στην περίπτωση που έχουμε «συνδέσει» πίνακα μαθημάτων (`scPAllCourses != 0`).
- Ενώ διασχίζουμε με τη **for** τον πίνακα κωδικών μαθημάτων aStCourses θα σταματήσουμε μόλις (αν) βρούμε τον πρώτο κωδικό που δεν υπάρχει στον πίνακα μαθημάτων.

Όπως στην *CourseCollection* έχουμε χάσει την πρόσβαση προς τις μεθόδους των στοιχείων (τύπου *Course*) του πίνακα έτσι και εδώ έχουμε χάσει την πρόσβαση προς τις μεθόδους των στοιχείων (τύπου *Student*) του πίνακα. Για να την ανακτήσουμε θα πρέπει να γράψουμε ενδιάμεσες συναρτήσεις. Τέτοιες συναρτήσεις θα πρέπει να γράψουμε τουλάχιστον για τις *add1Course()* και *delete1Course()*. Εδώ όμως χρειάζεται προσοχή: για κάθε στοιχείο του πίνακα sCourses υπάρχει ένα στοιχείο στον *Πίνακα Δηλώσεων Μαθημάτων*· έχουμε δηλαδή *πλεονασμό*. Και ένα πρόβλημα του πλεονασμού είναι το εξής: οι ενημερώσεις (εισαγωγές – διαγραφές) στους δύο πίνακες πρέπει να γίνονται με συνέπεια. Οι *add1Course()* και *delete1Course()* της *Student* δεν έχουν αυτην την ιδιότητα. Θα επανέλθουμε...

Prj04.6.1 Φύλαξη, Φόρτωση και Ευρετήριο

Κατ'αρχήν θα μπορούσαμε να πάρουμε τις *save()* και *load()* από αυτές της *CourseCollection* με «*find-and-replace*» αλλά μια νέα απαίτηση αυτής της δεύτερης εκδοχής του προβλήματος είναι:

> «*Για να ανακτήσουμε τη δυνατότητα κατ' ευθείαν πρόσβασης στα αντικείμενα του αρχείου θα χρειαστούμε ένα* **ευρετήριο** *(index) με στοιχεία τύπου*

```
struct IndexEntry
{
   unsigned int sIdNum;
   size_t       loc;
}; // IndexEntry
```

> *Το πρόγραμμά μας θα πρέπει να οργανώνει αυτά τα στοιχεία σε έναν πίνακα που θα τον φυλάγει σε ένα (μη-μορφοποιημένο) αρχείο* `students.ndx`.»

Να υλοποιήσουμε τα παραπάνω *μέσα* στη *StudentCollection;* Όχι! Ένα αντικείμενο της *StudentCollection* είναι ένα σύνολο που υλοποιείται στη μνήμη. Ο index είναι εργαλείο για τον χειρισμό μιας *StudentCollection* που έχει φυλαχθεί σε αρχείο.

Ο index δεν θα ανήκει σε αντικείμενο και θα έρχεται ως παράμετρος στη *save()*:

```
void StudentCollection::save( ofstream& bout, IndexEntry* index )
{
   if ( bout.fail() )
      throw StudentCollectionXptn( "save",
                                   StudentCollectionXptn::fileNotOpen );
   bout.write( reinterpret_cast<const char*>(&scNOfStudents),
               sizeof(scNOfStudents) );
   for ( int k(0); k < scNOfStudents; ++k )
   {
      index[k].sIdNum = scArr[k].getIdNum();
      index[k].loc = bout.tellp();
      scArr[k].save( bout );
   }
   if ( bout.fail() )
      throw StudentCollectionXptn( "save",
                                   StudentCollectionXptn::cannotWrite );
} // StudentCollection::save
```

Όταν έρχεται η σειρά για φύλαξη στο αρχείο του `scArr[k]` φυλάγουμε στο αντίστοιχο στοιχείο του ευρετηρίου τον αριθμό μητρώου (*sIdNum*) και τη θέση στο αρχείο (*loc*) όπου θα αρχίσει η αποθήκευση του `scArr[k]`. Έτσι, όταν τελειώσει η φύλαξη του `scArr` ο `index` «ξέρει» που αρχίζει η αποθήκευση του κάθε αντικειμένου-στοιχείου του `scArr`.

Η υπόλοιπη διαχείριση του `index` είναι υποχρέωση του προγράμματος-πελάτη της κλάσης.

Για τη φόρτωση δεν χρειάζεται το ευρετήριο:

```
void StudentCollection::load( ifstream& bin )
{
   StudentCollection tmp;
   unsigned int n;
   bin.read( reinterpret_cast<char*>(&n), sizeof(scNOfStudents) );
   if ( !bin.eof() )
   {
      for ( int k(0); k < n && !bin.fail(); ++k )
      {
         Student oneStudent;
         oneStudent.load( bin );
         tmp.insert1Student( oneStudent );
      }
      if ( bin.fail() )
         throw StudentCollectionXptn( "load",
                                      StudentCollectionXptn::cannotRead );
      swapArr( tmp );
   }
```

```
} // StudentCollection::load
```

Τι είναι η *swapArr()*; Μια μέθοδος που αντιμεταθέτει τους πίνακες δύο αντικειμένων *StudentCollection*:

```
void StudentCollection::swapArr( StudentCollection& rhs )
{
   std::swap( scArr, rhs.scArr );
   std::swap( scNOfStudents, rhs.scNOfStudents );
   std::swap( scReserved, rhs.scReserved );
} // StudentCollection::swapArr
```

Και γιατί δεν γράφουμε μια *swap()* όπως όλοι οι «κανονικοί» άνθρωποι (προγραμματιστές); Διότι μια τέτοια μέθοδος θα πρέπει να αντιμεταθέτει και τις τιμές των *scPAllCourses* πράγμα όχι και τόσο βολικό.[6]

Prj04.7 Η Κλάση *StudentInCourse*

Θα ξαναγράψουμε τη *StudentInCourse* με βάση τη συνταγή μας και παίρνοντας υπόψη μας τις αλλαγές που κάναμε στην *Course*:

```
class StudentInCourse
{
public:
// . . .
private:
   unsigned int      sicSIdNum;  // αριθμός μητρώου
   Course::CourseKey sicCCode;   // κωδικός μαθήματος
   float             sicMark;    // βαθμός στο μάθημα
}; // StudentInCourse
```

Δύο αντικείμενα αυτής της κλάσης, *a* και *b*, είναι ίσα αν και μόνον αν αναφέρονται στον ίδιο φοιτητή (*a.sicSIdNum* == *b.sicSIdNum*) για το ίδιο μάθημα (*a.sicCCode* == *b.sicCCode*). Δηλαδή, το ζεύγος (*sicSIdNum, sicCCode*) είναι *κλειδί*. Για χρήση στην κλάση εξαιρέσεων (αλλά και αλλού) ορίζουμε στην περιοχή "**public**" της κλάσης τον τύπο:

```
   struct SICKey
   {
      unsigned int      sIdNum;
      Course::CourseKey CCode;
      explicit SICKey( int aIdNum=0, string aCCode="" )
      {  sIdum = aIdNum;  CCode = Course::CourseKey(aCCode);  }
   }; // SICKey
```

και τη

```
   SICKey getKey() const
   {  return SICKey( sicSIdNum, sicCCode.s );  }
```

Για τον τύπο αυτόν μπορούμε να επιφορτώσουμε τον "!=":

```
bool operator!=( const StudentInCourse::SICKey& lhs,
                 const StudentInCourse::SICKey& rhs )
{  return ( (lhs.sIdNum != rhs.sIdNum) || (lhs.CCode != rhs.CCode) );  }
```

και στη συνέχεια για τον *StudentInCourse*:

```
bool operator!=( const StudentInCourse& lhs, const StudentInCourse& rhs )
{  return ( lhs.getKey() != rhs.getKey() );  }
```

Ας εφαρμόσουμε τώρα τη συνταγή μας.

AN: Η αναλλοίωτη της κλάσης θα είναι:

$$0 \texttt{ <= } sicSIdNum \texttt{ \&\&}$$
$$((strlen(sicCCode.s) \texttt{ == } Course\texttt{::}cCodeSz\text{-}1) \texttt{ || } (sicCCode.s) \texttt{ == } 0)) \texttt{ \&\&}$$
$$0 \texttt{ <= } sicMark \texttt{ <= } 10$$

[6] Κάτι δεν πάει καλά με τη σχεδίαση…

ΔΕ: Ο ερήμην δημιουργός (με αρχική τιμή) δηλώνεται ως:

```
   StudentInCourse( int aIdNum=0, string aCCode="" );
```

και ορίζεται:

```
StudentInCourse::StudentInCourse( int aIdNum, string aCCode )
{
   if ( aIdNum < 0 )
      throw StudentInCourseXptn( SICKey(), "StudentInCourse",
                                 StudentInCourseXptn::negIdNum, aIdNum );
   sicSIdNum = aIdNum;
   if ( aCCode.length() != Course::cCodeSz-1 &&
        aCCode.length() != 0 )
      throw StudentInCourseXptn( SICKey(sicSIdNum), "StudentInCourse",
                                 StudentInCourseXptn::keyLen, aCCode.c_str());
   sicCCode = Course::CourseKey( aCCode );
   sicMark = 0;
} // StudentInCourse::StudentInCourse
```

ΔΑ, ΤΕ, ΚΑ: Ο δημιουργός αντιγραφής και ο τελεστής εκχώρησης που μας δίνει ο μεταγλωττιστής κάνουν σωστά τις δουλειές τους. Πάντως, κατά τη συνήθειά μας, θα γράψουμε έναν κενό καταστροφέα:

```
   ~StudentInCourse() { };
```

GE: Οι μέθοδοι "*get*", εκτός από τη *getKey()* που είδαμε παραπάνω:

```
   unsigned int getSIdNum() const {  return sicSIdNum;  }
   const char* getCCode() const {  return sicCCode.s;  }
   float getMark() const {  return sicMark;  }
```

SE: Και οι μέθοδοι "*set*":

```
void StudentInCourse::setSIdNum( int aIdNum )
{
   if ( aIdNum <= 0 )
      throw StudentInCourseXptn( getKey(), "setSIdNum",
                                 StudentInCourseXptn::negIdNum, aIdNum );
   sicSIdNum = aIdNum;
} // StudentInCourse::setSIdNum

void StudentInCourse::setCCode( string aCCode)
{
   if ( aCCode.length() != Course::cCodeSz-1 )
      throw StudentInCourseXptn( getKey(), "setCCode",
                                 StudentInCourseXptn::keyLen, aCCode.c_str());
   sicCCode = Course::CourseKey( aCCode );
} // StudentInCourse::setCCode

void StudentInCourse::setMark( float aMark )
{
   if ( aMark < 0 || 10 < aMark )
      throw StudentInCourseXptn( getKey(), "setMark",
                                 StudentInCourseXptn::rangeError, aMark );
   sicMark = aMark;
} // StudentInCourse::setMark
```

Τη *setIdNumCCode()* που είχαμε θα την κρατήσουμε; Όχι, διότι, όπως πιθανότατα μαντεύεις, στη συνέχεια θα γράψουμε και μια κλάση *StudentInCourseCollection* όπου και θα γίνονται αυτά που έκανε η *setIdNumCCode()*.

Άλλες Μέθοδοι: Η *save()* είναι αυτή της αρχικής εκδοχής με μικρές προσαρμογές στις εξαιρέσεις και στη φύλαξη του κωδικού μαθήματος:

```
void StudentInCourse::save( ostream& bout ) const
{
   if ( bout.fail() )
      throw StudentInCourseXptn( getKey(), "save",
                                 StudentInCourseXptn::fileNotOpen );
   bout.write( reinterpret_cast<const char*>(&sicSIdNum), sizeof(sicSIdNum) );
   bout.write( sicCCode.s, Course::cCodeSz ); // κωδικός μαθήματος
```

```
   bout.write( reinterpret_cast<const char*>(&sicMark), sizeof(sicMark) );
   if ( bout.fail() )
      throw StudentInCourseXptn( getKey(), "save",
                                 StudentInCourseXptn::cannotWrite );
} // StudentInCourse::save
```

Για τη δεύτερη εφαρμογή θα χρειαστούμε και μια *load()*:

```
void StudentInCourse::load( istream& bin )
{
   StudentInCourse tmp;
   bin.read( reinterpret_cast<char*>(&tmp.sicSIdNum), sizeof(sicSIdNum) );
   if ( !bin.eof() )
   {
      bin.read( tmp.sicCCode.s, Course::cCodeSz );        // κωδικός μαθήματος
      bin.read( reinterpret_cast<char*>(&tmp.sicMark), sizeof(sicMark) );
      if ( bin.fail() )
         throw StudentInCourseXptn( getKey(), "load",
                                    StudentInCourseXptn::cannotWrite );
      *this = tmp;
   }
} // StudentInCourse::load
```

Η κλάση θα είναι:

```
class StudentInCourse
{
public:
   struct SICKey
   {
      unsigned int      sIdNum;
      Course::CourseKey CCode;
      explicit SICKey( int aIdNum=0, string aCCode="" )
      {  sIdNum = aIdNum;  CCode = Course::CourseKey(aCCode);  }
   }; // SICKey
   explicit StudentInCourse( int aIdNum=0, string aCCode="" );
   ~StudentInCourse() { };
// getters
   unsigned int getSIdNum() const {  return sicSIdNum;  }
   const char* getCCode() const {  return sicCCode.s;  }
   float getMark() const {  return sicMark;  }
   SICKey getKey() const
   {  return SICKey( sicSIdNum, sicCCode.s );  }
// setters
   void setSIdNum( int aIdNum );
   void setCCode( string aCode);
   void setMark( float aMark );
// other
   void save( ostream& bout ) const;
   void load( istream& bin );
private:
   unsigned int      sicSIdNum;  // αριθμός μητρώου
   Course::CourseKey sicCCode;   // κωδικός μαθήματος
   float             sicMark;    // βαθμός στο μάθημα
}; // StudentInCourse
```

Ακόμη, έχουμε επιφορτώσει τον "`!=`":

```
bool operator!=( const StudentInCourse::SICKey& lhs,
                 const StudentInCourse::SICKey& rhs )
{  return ( (lhs.sIdNum != rhs.sIdNum) || (lhs.CCode != rhs.CCode) );  }

bool operator!=( const StudentInCourse& lhs, const StudentInCourse& rhs )
{  return ( lhs.getKey() != rhs.getKey() );  }
```

Και η κλάση εξαιρέσεων:

```
struct StudentInCourseXptn
{
   enum { negIdNum, keyLen, rangeError, unknownCCode,
          fileNotOpen, cannotWrite };
   StudentInCourse::SICKey objKey;
```

```
   char                    funcName[100];
   int                     errorCode;
   char                    errStrVal[100];
   float                   errFltVal;
   StudentInCourseXptn( const StudentInCourse::SICKey obk,
                        const char* mn, int ec, const char* sv="" )
      :  objKey( obk ),  errorCode( ec )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';
      strncpy( errStrVal, sv, 99 );  errStrVal[99] = '\0';  }
   StudentInCourseXptn( const StudentInCourse::SICKey& obk,
                        const char* mn, int ec, float fv )
      :  objKey( obk ),  errorCode( ec ),  errFltVal( fv )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';  }
}; // StudentInCourseXptn
```

Prj04.8 Η Κλάση *StudentInCourseCollection*

Αν θυμηθούμε πώς γράψαμε τις *CourseCollection* και *StudentCollection* και πάρουμε υπόψη μας ότι «*το ζεύγος (sicSIdNum, sicCCode) είναι κλειδί*» γράφουμε μια πρώτη μορφή της

```
class StudentInCourseCollection
{
public:
   StudentInCourseCollection();
   ~StudentInCourseCollection()  {  delete[] siccArr;  };
// getters
   size_t getNOfStudentInCourses() const
   {  return siccNOfStudentInCourses;  }
   const StudentInCourse* getArr() const {  return siccArr;  }
// 1 StudentInCourse
   bool find1StudentInCourse( int aIdNum, string code ) const
   {  return ( findNdx(aIdNum, code) >= 0 );  }
   const StudentInCourse& get1StudentInCourse( int aIdNum,
                                               string code ) const;
   void delete1StudentInCourse( int aIdNum, string code );
   void add1StudentInCourse( const StudentInCourse& aStdnInCrs );
// other
   void save( ofstream& bout ) const;
   void load( ifstream& bin );
   void swapArr( StudentInCourseCollection& rhs );
   void display( ostream& tout ) const;
private:
   enum { siccIncr = 50 };
   StudentInCourse* siccArr;
   size_t           siccNOfStudentInCourses;
   size_t           siccReserved;

   StudentInCourseCollection( const StudentInCourseCollection& rhs ) { };
   StudentInCourseCollection& operator=(
                                  const StudentInCourseCollection& rhs ) { };

   void erase1StudentInCourse( int ndx );
   void insert1StudentInCourse( const StudentInCourse& aStdnInCrs );
   int findNdx( int aIdNum, string code ) const;
}; // StudentInCourseCollection
```

Ενδεικτικώς ας δούμε τη

```
int StudentInCourseCollection::findNdx( int aIdNum, string code ) const
{
   int ndx;
   if ( aIdNum <= 0 || code.length() != Course::cCodeSz-1 )
      ndx = -1;
   else
      ndx = linSearch( siccArr, siccNOfStudentInCourses,
                       0, siccNOfStudentInCourses-1,
                       StudentInCourse(aIdNum,code) );
```

```
   return ndx;
} // StudentInCourseCollection::findNdx
```

Εδώ πρόσεξε τα εξής:

- Αφού το κλειδί για τα αντικείμενα τύπου *StudentInCourse* είναι (*sicSIdNum*, *sicCCode*) η μέθοδος πρέπει να τροφοδοτηθεί με τα αντίστοιχα *aIdNum* και *code*. Τις ίδιες παραμέτρους έχουν και οι *find1StudentInCourse()*, *delete1StudentInCourse()*, *get1StudentInCourse()* και *delete1StudentInCourse()*.
- Πριν καλέσουμε τη *linSearch()* ελέγχουμε και τα δύο τμήματα του κλειδιού· θα πρέπει να έχουμε: *aIdNum* `> 0 &&` *code*`.`*length*`() ==` *Course*`::`*cCodeSz*`-1`.

Φυσικά, το ενδιαφέρον βρίσκεται στις *delete1StudentInCourse()* και *add1StudentInCourse()*. Η δεύτερη θα παίξει (και) τον ρόλο της *setIdNumCCode()* του Project 3.

Για να μπορεί η *add1StudentInCourse()* να παίξει τον ρόλο της *setIdNumCCode()* θα πρέπει να έχει πρόσβαση στον πίνακα μαθημάτων και στο μητρώο φοιτητών· θα βάλουμε και εδώ παραμέτρους για τους πίνακες; Όχι! Όπως κάναμε στη *StudentCollection* θα βάλουμε:

```
private:
// . . .
   StudentCollection* siccPAllStudents;
   CourseCollection*  siccPAllCourses;
```

με αρχικές τιμές στον δημιουργό:

```
StudentInCourseCollection::StudentInCourseCollection()
{
   try
   {
      siccReserved = siccIncr;
      siccArr = new StudentInCourse[ siccReserved ];
      siccNOfStudentInCourses = 0;
   }
   catch( bad_alloc& )
   {
      throw StudentInCourseCollectionXptn( "StudentInCourseCollection",
                                 StudentInCourseCollectionXptn::allocFailed );
   }
   siccPAllStudents = 0;
   siccPAllCourses = 0;
} // StudentInCourseCollection::StudentInCourseCollection
```

και –για τον χειρισμό τους– τις

```
public:
// . . .
   const StudentCollection* getPAllStudents() const
   {  return siccPAllStudents;  }
   const CourseCollection* getPAllCourses( ) const
   {  return siccPAllCourses;  }
// setters
   void setPAllStudents( const StudentCollection* pStudents )
   {  siccPAllStudents = pStudents;  }
   void setPAllCourses( const CourseCollection* pCourses )
   {  siccPAllCourses = pCourses;  }
```

Μετά απο'αυτά, αν στο πρόγραμμά μας έχουμε δηλώσει:

```
   CourseCollection allCourses;
   StudentCollection allStudents;
   StudentInCourseCollection allEnrollments;
```

μπορούμε να δώσουμε:

```
   allEnrollments.setPAllStudents( &allStudents );
   allEnrollments.setPAllCourses( &allCourses );
```

Η *delete1StudentInCourse()* και *add1StudentInCourse()* θα κάνουν ελέγχους –και ενημερώσεις– αν βρίσκουν `siccPAllStudents != 0` ή/και `siccPAllCourses != 0` για να ανταποκριθούν στην απαίτηση: «*οι ενημερώσεις (εισαγωγές – διαγραφές) στους δύο πίνακες*

[sCourses της *Student* και siccArr της *StudentInCourseCollection*] *πρέπει να γίνονται με συνέπεια.*»

Ας δούμε λοιπόν ένα σχέδιο για την *add1StudentInCourse()*:

- Πριν από οποιαδήποτε ενέργεια θα πρέπει να ελέγξουμε αν υπάρχει ήδη καταχωρισμένη η εγγραφή του φοιτητή στο συγκεκριμένο μάθημα. Αν υπάρχει δεν υπάρχει δουλειά για τη μέθοδο:

```
void add1StudentInCourse( const StudentInCourse& aStdnInCrs )
{
   if ( στον siccArr δεν υπάρχει στοιχείο με
                              (aStdnInCrs.getSIdNum(),aStdnInCrs.getCCode()) )
   {
   // . . .
   }
} // StudentInCourseCollection::add1StudentInCourse
```

- Για να προχωρήσουμε στην ενημέρωση του πίνακα (εισαγωγή στοιχείου) θα πρέπει να έχουμε σιγουρέψει ότι πρόκειται για εγγραφή υπαρκτού φοιτητή σε υπαρκτό μάθημα:

```
   if ( siccPAllCourses == 0 )
      throw …
   if ( δεν υπάρχει μάθημα με κωδικό aStdnInCrs.getCCode() )
      throw …
   if ( siccPAllStudents == 0 )
      throw …
   if ( δεν υπάρχει φοιτητής με α.μ. aStdnInCrs.getSIdNum() )
      throw …
```

- Αν βρούμε όλα όσα χρειαζόμαστε κάνουμε τις ενημερώσεις:

```
   Ενημέρωσε το μάθημα (αριθμός φοιτητών)
   Πάρε το αντικείμενο του μαθήματος
   Ενημέρωσε τον φοιτητή (εισαγωγή μαθήματος)
   Κάνε εισαγωγή του aStdnInCrs στον siccArr
```

Για να ενημερώσουμε το μάθημα πρέπει να καλέσουμε την *add1Student()* της συλλογής (*CourseCollection*) που δείχνει το *siccPAllCourses*:

```
      siccPAllCourses->add1Student( aStdnInCrs.getCCode() );
```

Από την ίδια συλλογή παίρνουμε το αντικείμενο του μαθήματος για να το χρησιμοποιήσουμε στην ενημέρωση το αντικείμενο του φοιτητή:

```
      Course oneCourse( siccPAllCourses->get1Course(aStdnInCrs.getCCode()) );
```

Για να ενημερώσουμε το αντικείμενο του φοιτητή πρέπει να καλέσουμε την *add1Course* της συλλογής (*StudentCollection*) που δείχνει το *siccPAllStudents*:[7]

```
      siccPAllStudents->add1Course( aStdnInCrs.getSIdNum(), oneCourse );
```

Μετά από αυτά εισάγουμε το *aStdnInCrs* στον πίνακα siccArr της συλλογής *StudentInCourseCollection*:

```
      insert1StudentInCourse( aStdnInCrs );
```

Να ολόκληρη η μέθοδος:

```
void StudentInCourseCollection::add1StudentInCourse(
                                          const StudentInCourse& aStdnInCrs )
{
   if ( findNdx(aStdnInCrs.getSIdNum(), aStdnInCrs.getCCode()) < 0 )
   { // δεν υπάρχει στοιχείο
      if ( siccPAllCourses == 0 )
         throw StudentInCourseCollectionXptn( "add1StudentInCourse",
                                    StudentInCourseCollectionXptn::noCrs );
```

[7] Δεν είναι απαραίτητο να χρησιμοποιήσουμε την τοπική μεταβλητή *oneCourse*· θα μπορούσαμε να γράψουμε:

```
      pAllStudents->add1Course( aStdnInCrs.getSIdNum(),
                          pAllCourses->get1Course( aStdnInCrs.getCCode() );
```

```
      if ( !(siccPAllCourses->find1Course(aStdnInCrs.getCCode())) )
         throw StudentInCourseCollectionXptn( "add1StudentInCourse",
                                  StudentInCourseCollectionXptn::unknownCrs,
                                              aStdnInCrs.getCCode() );
      if ( siccPAllStudents == 0 )
         throw StudentInCourseCollectionXptn( "add1StudentInCourse",
                                   StudentInCourseCollectionXptn::noStdnt );
      if ( !(siccPAllStudents->find1Student(aStdnInCrs.getSIdNum())) )
         throw StudentInCourseCollectionXptn( "add1StudentInCourse",
                                 StudentInCourseCollectionXptn::unknownStdn,
                                              aStdnInCrs.getSIdNum() );
      siccPAllCourses->add1Student( aStdnInCrs.getCCode() );
      Course oneCourse( siccPAllCourses->get1Course(aStdnInCrs.getCCode()) );
      siccPAllStudents->add1Course( aStdnInCrs.getSIdNum(), oneCourse );
      insert1StudentInCourse( aStdnInCrs );
   }
} // StudentInCourseCollection::add1StudentInCourse
```

Η *delete1StudentInCourse()* είναι κάπως πιο απλή:

```
void StudentInCourseCollection::delete1StudentInCourse( int aIdNum,
                                                         string code )
{
   int ndx( findNdx( aIdNum, code );
   if ( ndx >= 0 ) // υπάρχει στοιχείο
   {
      if ( siccPAllCourses == 0 )
         throw StudentInCourseCollectionXptn( "delete1StudentInCourse",
                                       StudentInCourseCollectionXptn::noCrs );
      if ( siccPAllStudents == 0 )
         throw StudentInCourseCollectionXptn( "delete1StudentInCourse",
                                      StudentInCourseCollectionXptn::noStdnt );
      siccPAllCourses->delete1Student( code );
      Course oneCourse( siccPAllCourses->get1Course(code) );
      siccPAllStudents->delete1Course( aIdNum, oneCourse );
      erase1StudentInCourse( ndx );
   }
} // StudentInCourseCollection::delete1StudentInCourse
```

Η *StudentInCourseCollection* γίνεται τελικώς:

```
class CourseCollection;
class StudentCollection;

class StudentInCourseCollection
{ // version 1
public:
   StudentInCourseCollection();
   ~StudentInCourseCollection()  {  delete[] siccArr;  };
// getters
   size_t getNOfStudentInCourses() const
   {  return siccNOfStudentInCourses;  }
   const StudentInCourse* getArr() const {  return siccArr;  }
   const StudentCollection* getPAllStudents() const
   {  return siccPAllStudents;  }
   const CourseCollection* getPAllCourses( ) const
   {  return siccPAllCourses;  }
// setters
   void setPAllStudents( StudentCollection* pStudents )
   {  siccPAllStudents = pStudents;  }
   void setPAllCourses( CourseCollection* pCourses )
   {  siccPAllCourses = pCourses;  }
// 1 StudentInCourse
   bool find1StudentInCourse( int aIdNum, string code ) const
   {  return ( findNdx(aIdNum, code) >= 0 );  }
   void delete1StudentInCourse( int aIdNum, string code );
   void add1StudentInCourse( const StudentInCourse& aStdnInCrs );
   const StudentInCourse& get1StudentInCourse( int aIdNum,
                                               string code ) const;
```

```
// other
   void save( ofstream& bout ) const;
   void load( ifstream& bin );
   void swapArr( StudentInCourseCollection& rhs );
   void display( ostream& tout ) const;
private:
   enum { siccIncr = 50 };
   StudentInCourse*   siccArr;
   size_t             siccNOfStudentInCourses;
   size_t             siccReserved;
   StudentCollection* siccPAllStudents;
   CourseCollection*  siccPAllCourses;

   StudentInCourseCollection( const StudentInCourseCollection& rhs ) { };
   StudentInCourseCollection& operator=(
                                  const StudentInCourseCollection& rhs ) { };
   void erase1StudentInCourse( int ndx );
   void insert1StudentInCourse( const StudentInCourse& aStdnInCrs );
   int findNdx( int aIdNum, string code ) const;
}; // StudentInCourseCollection
```

Πρόσεξε τις δύο *προειδοποιήσεις δήλωσης* για τις κλάσεις *CourseCollection* και *StudentCollection*. Είναι απαραίτητες για να γίνουν δεκτές από τον μεταγλωττιστή οι δηλώσεις

```
   StudentCollection* siccPAllStudents;
   CourseCollection*  siccPAllCourses;
```

Η αντίστοιχη κλάση εξαιρέσεων:

```
struct StudentInCourseCollectionXptn
{ // version 1
   enum { allocFailed, notFound, noCrs, unknownCrs, noStdnt,
          unknownStdn, fileNotOpen, cannotWrite, cannotRead };
   char funcName[100];
   int  errorCode;
   char errStrVal[100];
   char errIntVal;
   StudentInCourse::SICKey errSICVal;
   StudentInCourseCollectionXptn( const char* mn, int ec, const char* sv="" )
      :  errorCode( ec )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';
      strncpy( errStrVal, sv, 99 );  errStrVal[99] = '\0';  };
   StudentInCourseCollectionXptn( const char* mn, int ec, int iv )
      :  errorCode( ec ),  errIntVal( iv )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';  };
   StudentInCourseCollectionXptn( const char* mn, int ec,
                                  const StudentInCourse::SICKey& sk )
      :  errorCode( ec ),  errSICVal( sk )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';  };
}; // StudentInCourseCollectionXptn
```

Prj04.9 Πώς θα Γίνονται οι Ενημερώσεις

Ας πούμε ότι θέλουμε να κάνουμε εισαγωγή της εγγραφής ενός φοιτητή σε κάποιο μάθημα· θα πρέπει

- Να ενημερώσουμε ένα μέλος του πίνακα μαθημάτων (τύπου *Course Collection*), με την *add1Student()*.
- Να ενημερώσουμε ένα μέλος του μητρώου φοιτητών (τύπου *StudentCollection*), με την *add1Course()*.
- Να εισαγάγουμε ένα νέο στοιχείο στον πίνακα εγγραφών στα μαθήματα (τύπου *StudentInCourseCollection*) με την *add1StudentInCourse()*.

Ποιος διασφαλίζει ότι όλα αυτά θα γίνουν σωστά ώστε να είναι σωστό το περιεχόμενο των πινάκων μας; Το πρόγραμμα που τους χρησιμοποιεί; Ούτε για αστείο! Αυτό πρέπει να διασφαλίζεται από αυτόν που γράφει τις κλάσεις. *Εδώ* θα γίνει.

Η *StudentInCourseCollection*::*add1StudentInCourse()*, όπως τη γράψαμε, κάνει όλες τις ενημερώσεις που παραθέτουμε παραπάνω. Αυτό όμως δεν διασφαλίζει οτιδήποτε, αφού ο προγραμματιστής μπορεί να βάλει κλήσεις (και) προς τις άλλες μεθόδους. Πώς διορθώνεται αυτό; Έχουμε δύο επιλογές:

- Να γράψουμε τις (*Course Collection*::)*add1Student()*, (*StudentCollection*::)*add1Course()* έτσι ώστε να κάνουν όλες τις ενημερώσεις. Με αυτήν την επιλογή θα έχεις το δικαίωμα να κάνεις εισαγωγή εγγραφής σε ένα μάθημα με οποιαδήποτε από τις τρεις μεθόδους. Αλλά και κάθε φορά που θα αλλάζεις το λογισμικό σου θα πρέπει να αλλάζεις και τις τρεις μεθόδους με συνεπή τρόπο.
- Να κρύψουμε (σε περιοχή **private**) τις (*CourseCollection*::)*add1Student()*, (*StudentCollection*::)*add1Course()* ώστε η *add1StudentInCourse()* να είναι η μόνη υπεύθυνη για όλες τις ενημερώσεις.

Η δεύτερη επιλογή έχει αυτό το πολύ σημαντικό πλεονέκτημα: μια μεθοδος «*είναι η μόνη υπεύθυνη για όλες τις ενημερώσεις*.» Η πρώτη επιλογή, χωρίς αυτό το «προσόν», σου εγγυάται(!) ότι –συν τω χρόνω, με τις αλλαγές και προσαρμογές που θα πρέπει να γίνονται– θα έχεις λογισμικό και δεδομένα με πολλά προβλήματα.

Φυσικά, θα πρέπει να έχουμε και *μια μονον μέθοδο υπεύθυνη για τη όλες τις ενημερώσεις που αφορούν τη διαγραφή* ενός φοιτητή από ένα μάθημα.

Πώς υλοποιείται η επιλογή μας;

- Δηλώνουμε στην *CourseCollection* και στην *StudentCollection*:

```
friend void StudentInCourseCollection::add1StudentInCourse(
                                        const StudentInCourse& aStdnInCrs );
friend void StudentInCourseCollection::delete1StudentInCourse( int aIdNum,
                                                               string code );
```

- «Κρύβουμε» στην περιοχή **private** της *CourseCollection* τις *add1Student* και *delete1Student*.
- «Κρύβουμε» στην περιοχή **private** της *StudentCollection* τις *add1Course* και *delete1Course*.

Prj04.10 Οι Άλλες Συλλογές Τελικώς

Τώρα μπορούμε να δώσουμε τις τελικές –αν και όχι πλήρεις– μορφές των κλάσεων *CourseCollection* και *StudentCollection*.

Prj04.10.1 Η Κλάση *CourseCollection*

Θα ξαναδούμε τώρα την *CourseCollection*::*delete1Course()*. Η λύση που δώσαμε στην §Prj04.3.1 φαίνεται κάπως «τεμπέλικη» διότι αυτό που λέμε στην πραγματικότητα είναι το εξής: Θέλεις να διαγράψεις ένα μάθημα στο οποίο είναι γραμμένοι φοιτητές (*cNoOfStudents* > 0); Τότε:

- Διάγραψε πρώτα μια προς μια τις εγγραφές των φοιτητών στο συγκεκριμένο μάθημα ενημερώνοντας παραλλήλως τα αντικείμενα των φοιτητών.
- Μετά από αυτό (αφού θα έχεις *cNoOfStudents* == 0) διάγραψε και το αντικείμενο του μαθήματος.

Παίρνοντας υπόψη μας και αυτά που λέγαμε στην Prj04.4 σου προτείνουμε (ως άσκηση) το εξής:

- Εφοδίασε το σύστημά σου με έναν πίνακα τύπου *DeletedCourseCollection* και έναν *DeletedStudentInCourseCollection*[8] με στοιχεία αντίστοιχων κλάσεων

```
class DeletedCourse
{
private:
// . . .
public:
   Course dcCourse;
   time_t dcDelTime; // ή DateTime dcDelTime (χρόνος διαγραφής)
}; // DeletedCourse
class DeletedStudentInCourse
{
private:
// . . .
public:
   StudentInCourse dsiccEnroll;
   time_t          dsiccDelTime; // ή DateTime dsiccDelTime
}; // DeletedStudentInCourse
```

- Μην πειράξεις την *CourseCollection*::*delete1Course()* αλλά γράψε μιαν άλλη μέθοδο, ας την πούμε *CourseCollection*::*moveOut1Course()*, που θα κάνει τα εξής:
  - Θα διαγράφει τις εγγραφές στο μάθημα από τον πίνακα δηλώσεων και αφού εξοπλίσει την κάθε μια με τον χρόνο διαγραφής (μετατροπή σε αντικείμενο *DeletedStudentInCourse*) θα τις εισάγει στον πίνακα τύπου *DeletedStudentInCourseCollection*. Προφανώς θα χρειαστείς και μια *StudentInCourseCollection*::*moveOut1StudentInCourse()*.
  - Θα διαγράφει το αντικείμενο του μαθήματος από τον πίνακα μαθημάτων και αφού το μετατρέψει σε αντικείμενο *DeletedCourse* θα το εισάγει στον πίνακα *DeletedCourseCollection*.
  - Φυσικά, θα πρέπει να ενημερώνει και τα αντικείμενα των φοιτητών που έχουν εγγραφεί στο μάθημα που διαγράφεται.

Όλες οι διαγραφές από τον πίνακα δηλώσεων θα πρέπει να γίνονται με τη *StudentInCourseCollection*::*delete1StudentInCourse()* και μόνον. Για να έχεις πρόσβαση από τον πίνακα μαθημάτων στον πίνακα των εγγραφών στα μαθήματα θα χρειαστείς ένα βέλος:

```
   StudentInCourseCollection* ccPAllEnrollments;
```

που θα παίρνει τιμή "`0`" όταν δημιουργείται η συλλογή (πίνακας μαθημάτων) και θα το διαχειρίζεσαι με τις:

```
const StudentInCourseCollection*
                        CourseCollection::getPAllEnrollments() const
{  return ccPAllEnrollments;  }

void CourseCollection::setPAllEnrollments(
                              StudentInCourseCollection* pEnrollments )
{  ccPAllEnrollments = pEnrollments;  }
```

Αφού σχεδιάσεις και υλοποιήσεις τους πίνακες διαγραφέντων μάλλον θα πρέπει να βάλεις αντίστοιχα εργαλεία και για αυτούς.

Ο ορισμός της κλάσης *CourseCollection* είναι:

```
class CourseCollection
{  // version 1
friend void StudentInCourseCollection::add1StudentInCourse(
                                        const StudentInCourse& aStdnInCrs );
friend void StudentInCourseCollection::delete1StudentInCourse( int aIdNum,
                                                               string code );
public:
   CourseCollection();
```

[8] Φυσικά θα χρειαστεί και ένας πίνακας *DeletedStudentCollection*.

```
   ~CourseCollection()  {  delete[] ccArr;  };
// getters
   size_t getNOfCourses() const {  return ccNOfCourses;  }
   const Course* getArr() const {  return ccArr;  }
//   const StudentInCourseCollection* getPAllEnrollments() const
//   {  return ccPAllEnrollments;  }
// setters
//   void setPAllEnrollments( StudentInCourseCollection*
//                                                    pEnrollments)
//   {  ccPAllEnrollments = pEnrollments;  }
// 1 Course
   bool find1Course( const string& code ) const
   {  return ( findNdx(code) >= 0 );  };
   void delete1Course( string code );
   void add1Course( const Course& aCourse );
   const Course& get1Course( string code ) const;
// other
   void save( ofstream& bout ) const;
   void load( ifstream& bin );
   void swap( CourseCollection& rhs );
   void display( ostream& tout ) const;
private:
   enum { ccIncr = 30 };
   Course*                   ccArr;
   size_t                    ccNOfCourses;
   size_t                    ccReserved;
//   StudentInCourseCollection* ccPAllEnrollments;

   CourseCollection( const CourseCollection& rhs ) { };
   CourseCollection& operator=( const CourseCollection& rhs ) { };
   void erase1Course( int ndx );
   void insert1Course( const Course& aCourse );
   void add1Student( string code );
   void delete1Student( string code );
   int findNdx( const string& code ) const;
}; // CourseCollection
```

Σε σχόλια έχουμε βάλει τα εργαλεία για την πρόσβαση στον πίνακα εγγραφών στα μαθήματα.[9]

Η αντίστοιχη κλάση εξαιρέσεων:

```
struct CourseCollectionXptn
{  // version 1
   enum { allocFailed, notFound, entity, cannotDel, prereqRef,
          noEnroll, enrollRef, fileNotOpen, cannotWrite,
          cannotRead };
   char funcName[100];
   int  errorCode;
   char errStrVal[100];
   int  errIntVal;
   CourseCollectionXptn( const char* mn, int ec,
                         const char* sv="", int iv=0 )
```

[9] Και πώς θα δεχθεί ο μεταγλωττιστής τη δήλωση της *ccPAllEnrollments*; Δεν θα χρειαστεί μια προειδοποίηση δήλωσης της *StudentInCourseCollection* πριν από τη δήλωση της `CourseCollection`; Εδώ έχουμε και άλλες απαιτήσεις από τις δηλώσεις "`friend`" των `StudentInCourseCollection::add1StudentInCourse()` και `StudentInCourseCollection::delete1StudentInCourse()`. Εδώ το πρόβλημα λύνεται με τη δήλωση της *StudentInCourseCollection* πριν από τη δήλωση της *CourseCollection*. Αφού παρόμοια ισχύουν και για την *StudentCollection*, στο πρόγραμμά μας θα πρέπει να βάλουμε:

```
#include "StudentInCourseCollection.h"
#include "CourseCollection.h"
#include "StudentCollection.h"
```

Έτσι, όταν έρχεται η δήλωση της *CourseCollection* (και μετά, της *Student Collection*) ο μεταγλωττιστής έχει ήδη βρει τη *StudentInCourseCollection* και τις μεθόδους της *add1StudentInCourse()* και *delete1StudentInCourse()*.

```
      :  errorCode( ec ),  errIntVal( iv )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';
      strncpy( errStrVal, sv, 99 );  errStrVal[99] = '\0';  };
}; // CourseCollectionXptn
```

Παρατήρηση: ▶

Κρύψαμε τον δημιουργό αντιγραφής και τον τελεστή εκχώρησης για να εμποδίσουμε την αντιγραφή αντικειμένων κλάσης *CourseCollection*. Τελικώς, απλώς τη δυσκολέψαμε: με τις *getNOfCourses()*, *getArr()* και *add1Course()* μπορείς να δημιουργήσεις αντίγραφα. Πάντως, έχουμε την εξής ελπίδα: ο προγραμματιστής που θα προσπαθήσει να δημιουργήσει αντίγραφα αντικειμένων κλάσης *CourseCollection*, θα δει τα αχρηστευμένα και θα το ξανασκεφτεί...◀

Prj04.10.2 Η Κλάση *StudentCollection*

Το πρώτο πράγμα που θα πρέπει να κάνουμε είναι η διόρθωση της *add1Student()*. Αφού είπαμε ότι η εισαγωγή εγγραφών σε μαθήματα θα γίνεται από «μια πόρτα», την *add1StudentInCourse()*, αυτή θα πρέπει να καλούμε στη **for** της *add1Student()*. Ο έλεγχος

```
if ( !scPAllCourses->find1Course(aStCourses[k].s) )
```

είναι περιττός αφού γίνεται από την *add1StudentInCourse()*:

```
void StudentCollection::add1Student( const Student& aStudent )
{
   int ndx( findNdx(aStudent.getIdNum()) );
   if ( ndx < 0 ) // δεν βρέθηκε το κλειδί
   {
      if ( scPAllEnrollments == 0 )
         throw StudentCollectionXptn( "add1Student",
                                      StudentCollectionXptn::noEnroll );
      insert1Student( aStudent );
      const Course::CourseKey* aStCourses( aStudent.getCourses() );
      for ( int k(0); k < aStudent.getNoOfCourses(); ++k )
      {
         scPAllEnrollments->add1StudentInCourse(
                                 StudentInCourse(aStudent.getIdNum(),
                                                 aStCourses[k].s) );
      }
   }
} // StudentCollection::add1Student
```

Πρόσεξε τα εξής:

- Θα χρειαστούμε ένα βέλος –ας το πούμε *scPAllEnrollments*– προς τον *Πίνακα Δηλώσεων Μαθημάτων*. Η τιμή του ελέγχεται στη δεύτερη **if**. *Οι δηλώσεις και οι μέθοδοι χειρισμού του βέλους προς τον πίνακα μαθημάτων θα αντικατασταθούν από τα αντίστοιχα του scPAllEnrollments*.
- Με την *insert1Student()* εισάγεται και ο πίνακας με τους κωδικούς μαθημάτων. Στη συνέχεια, η *StudentInCourseCollection*::*add1StudentInCourse()* θα καλεί τη *StudentCollection*::*add1Course()* που καλεί τη *Student*::*add1Course()* για να εισαγάγει τον κάθε κωδικό στον πίνακα κωδικών μαθημάτων. Αλλά η τελευταία δεν κάνει εισαγωγή αν τον βρει ήδη στον πίνακα.
- Υπάρχει μια σοβαρή διαφορά της *StudentCollection*::*add1Student()* από την *CourseCollection*::*add1Course()*:
  - Στο μητρώο φοιτητών εισάγουμε οποιοδήποτε αντικείμενο κλάσης *Student* χωρίς οποιοδήποτε πρόβλημα. Και αυτό διότι το αντικείμενο *Student*, με τον πίνακα μαθημάτων που περιέχει, μας επιτρέπει να ενημερώσουμε σωστά τον πίνακα δηλώσεων μαθημάτων και τον πίνακα μαθημάτων.
  - Στον πίνακα μαθημάτων εισάγουμε αντικείμενα κλάσης *Course* με *μηδενισμένο το ocNoOfStudents*. Γιατί; Διότι ένα αντικείμενο *Course* έχει *μόνον το πλήθος* των

φοιτητών που το ζήτησαν και *όχι τους αριθμούς μητρώου*. Έτσι είναι αδύνατη η ενημέρωση του πίνακα δηλώσεων μαθημάτων και των αντικειμένων με τα στοιχεία των φοιτητών.

Ας έλθουμε τώρα στη *StudentCollection*::*delete1Student()*, που την κάναμε «τεμπέλικη», αφού σβήνει μόνον φοιτητές που δεν είναι γραμμένοι σε μαθήματα. Παρ' όλο που η διαγραφή των εγγραφών ενός φοιτητή σε 4 ή 5 μαθήματα από τον *Πίνακα Δηλώσεων Μαθημάτων* δεν είναι τόσο «τραγική» όσο η διαγραφή των εγγραφών μερικών δεκάδων φοιτητών σε ένα μάθημα, θα σου προτείνουμε να χειριστείς τη διαγραφή φοιτητή με παρόμοιο τρόπο:

- Όρισε μια κλάση:

```
class DeletedStudent
{
private:
// . . .
public:
   Student dsStudent;
   time_t  dsDelTime; // ή DateTime dsDelTime (χρόνος διαγραφής)
}; // DeletedStudent
```

- Μην πειράξεις την («τεμπέλικη») *StudentCollection*::*delete1Student()* αλλά γράψε μιαν άλλη μέθοδο, ας την πούμε *StudentCollection*::*moveOut1Student()*, που θα κάνει τα εξής:
  - Θα διαγράφει τις εγγραφές του φοιτητή στα μαθήματα από τον πίνακα δηλώσεων και αφού εξοπλίσει την κάθε μια με τον χρόνο διαγραφής (μετατροπή σε αντικείμενο *DeletedStudentInCourse*) θα τις εισάγει στον πίνακα τύπου *DeletedStudentInCourseCollection* που είδαμε πιο πριν.
  - Θα διαγράφει το αντικείμενο του φοιτητή από το μητρώο φοιτητών και αφού το μετατρέψει σε αντικείμενο *DeletedStudent* θα το εισάγει στον πίνακα *DeletedStudentCollection*.
  - Θα πρέπει να ενημερώνει και τα αντικείμενα των μαθημάτων στα οποία είχε εγγραφεί ο φοιτητής που διαγράφεται.

Θα πρέπει ακόμη να γράψουμε τις «ενδιάμεσες» μεθόδους *add1Course()* και *delete1Course()*. Να θυμίσουμε ότι θα είναι *εσωτερικές* και θα καλούντσι μόνον από τις *φίλες* *StudentInCourseCollection*::*add1StudentInCourse()* και *StudentInCourseCollection*::*delete1StudentInCourse()*.

```
void StudentCollection::add1Course( int aIdNum, const Course& aCourse )
{
   int ndx( findNdx(aIdNum) );
   if ( ndx < 0 )
      throw StudentCollectionXptn( "add1Course",
                                   StudentCollectionXptn::notFound, aIdNum );
   scArr[ndx].add1Course( aCourse );
} // StudentCollection::add1Course

void StudentCollection::delete1Course( int aIdNum, const Course& aCourse )
{
   int ndx( findNdx(aIdNum) );
   if ( ndx < 0 )
      throw StudentCollectionXptn( "delete1Course",
                                   StudentCollectionXptn::notFound, aIdNum );
   scArr[ndx].delete1Course( aCourse );
} // StudentCollection::add1Course
```

Να πώς δηλώνεται η κλάση *StudentCollection*:

```
class StudentCollection
{  // version 1
friend void StudentInCourseCollection::add1StudentInCourse(
                                 const StudentInCourse& aStdnInCrs );
friend void StudentInCourseCollection::delete1StudentInCourse(
```

```
                                        int aIdNum, string code );
public:
   StudentCollection();
   ~StudentCollection()  {  delete[] scArr;  delete[] scIndex;  };
// getters
   size_t getNOfStudents() const {  return scNOfStudents;  }
   const Student* getArr() const {  return scArr;  }
   const StudentInCourseCollection* getPAllEnrollments() const
   {  return scPAllEnrollments;  }
// setters
   void setPAllEnrollments( StudentInCourseCollection*
                                                    pEnrollments )
   {  scPAllEnrollments = pEnrollments;  }
// 1 Student
   bool find1Student( int aIdNum ) const
   {  return ( findNdx(aIdNum) >= 0 );  };
   const Student& get1Student( int aIdNum ) const;
   void add1Student( const Student& aStudent );
   void delete1Student( int aIdNum );
// other
   void save( ofstream& bout, IndexEntry* index );
   void load( ifstream& bin );
   void swapArr( StudentCollection& rhs );
   void display( ostream& tout ) const;
private:
   enum { scIncr = 30 };
   Student*                   scArr;
   size_t                     scNOfStudents;
   size_t                     scReserved;
   StudentInCourseCollection* scPAllEnrollments;

   StudentCollection( const StudentCollection& rhs ) { };
   StudentCollection& operator=( const StudentCollection& rhs ) { };
   void erase1Student( int ndx );
   void insert1Student( const Student& aStudent );
   void add1Course( int aIdNum, const Course& aCourse );
   void delete1Course( int aIdNum, const Course& aCourse );
   int findNdx( int aIdNum ) const;
}; // StudentCollection
```

Η αντίστοιχη κλάση εξαιρέσεων:

```
struct StudentCollectionXptn
{  // version 1
   enum { allocFailed, notFound, noEnroll, enrollRef,
          fileNotOpen, cannotWrite, cannotRead };
   char funcName[100];
   int  errorCode;
   char errStrVal[100];
   char errIntVal;
   StudentCollectionXptn( const char* mn, int ec,
                          const char* sv="", int iv=0  )
      :  errorCode( ec ),  errIntVal( iv )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';
      strncpy( errStrVal, sv, 99 );  errStrVal[99] = '\0';  };
   StudentCollectionXptn( const char* mn, int ec, int iv )
      :  errorCode( ec ),  errIntVal( iv )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';  };
}; // StudentCollectionXptn
```

Prj04.11 Το 1ο Πρόγραμμα – Δημιουργία

Κρατούμε κατά βάση το σχέδιο που χρησιμοποιήσαμε στην πρώτη εκδοχή του προγράμματος (Prj03.2)· η υλοποίηση όμως θα είναι διαφορετική και μάλλον πιο απλή αφού αρκετή πολυπλοκότητα «κρύφτηκε» μέσα στις κλάσεις και στις μεθόδους τους.

Ας ξεκινήσουμε με τη

```
void loadCourses( string flNm, CourseCollection& allCourses )
{
// Άνοιξε το αρχείο
   ifstream bin( flNm.c_str(), ios_base::binary );
   if ( bin.fail() )
      throw ProgXptn( "loadCourses", ProgXptn::cannotOpen,
                      flNm.c_str() );
// Τώρα διάβαζε
   Course oneCourse;
   loadSylCourse( bin, oneCourse );
   while ( !bin.eof() )
   {
      oneCourse.clearStudents();
      allCourses.add1Course( oneCourse );
      loadSylCourse( bin, oneCourse );
   } // while
   bin.close();
} // loadCourses
```

(όπου *loadSylCourse()* είναι –χωρίς αλλαγές– αυτή που είδαμε στο πρόγραμμα του Project 3.)

Εδώ βλέπεις τι εννοούμε: όλη η διαχείριση δυναμικής μνήμης κρύφτηκε στις μεθόδους της *allCourses*.

Prj04.11.1 Αρχείο Φοιτητών και Δηλώσεων Μαθημάτων

Ας δούμε τώρα πώς θα διαβάσουμε το αρχείο `enrllmnt.txt`:

- Μπορούμε να διαβάσουμε *όλα* τα στοιχεία (μαζί με τους κωδικούς μαθημάτων που δήλωσε) που αφορούν έναν φοιτητή και να τα εισαγάγουμε στο Μητρώο Φοιτητών αν δεν υπάρχει ήδη μέσα εγγραφή για τον συγκεκριμένο φοιτητή.
- Μπορούμε να διαβάσουμε τα στοιχεία του φοιτητή –*αλλά όχι τους κωδικούς των μαθημάτων*– και να τα εισαγάγουμε στο Μητρώο Φοιτητών. Στη συνέχεια διαβάζουμε και εισάγουμε –στο αρχείο δηλώσεων– έναν προς έναν τους κωδικούς μαθημάτων μαζί με τον αριθμό μητρώου του φοιτητή.

  Ας δούμε τι θα γίνεται με τα λάθη στην *πρώτη περίπτωση*:

- Αν στο Μητρώο Φοιτητών βρεθεί ο αριθμός μητρώου δεν γίνεται η εισαγωγή και προχωρούμε στον επόμενο φοιτητή.
- Αν στο αντικείμενο του φοιτητή βρεθεί κωδικός μαθήματος που δεν υπάρχει στον πίνακα μαθημάτων θα πάρουμε από την *add1StudentInCourse()* –μέσω της *add1Student()*– εξαίρεση. Στην περίπτωση αυτή θα πρέπει να διαγράψουμε τον κωδικό από το αντικείμενο του φοιτητή και να δοκιμάσουμε πάλι την εισαγωγή. Αν πάρουμε και πάλι *unknownCrs* διαγράφουμε τον κωδικό και ξαναδοκιμάζουμε. Αυτή η διαδικασία επαναλαμβάνεται μέχρις ότου γίνει η εισαγωγή. Για να καταλάβεις καλύτερα αυτό που γίνεται, ας πούμε ότι έχουμε δηλώσεις 6 (0..5) μαθημάτων και έχουμε λάθος στους κωδικούς 2, 3 και 5.
  - Την πρώτη φορά: γίνονται αναζητήσεις για τους κωδικούς 0, 1 και 2. Διαγράφουμε τον κωδικό στη θέση 2 και εκεί έρχεται αυτός που ήταν στη θέση 5. Ο πίνακας μένει με 5 (0..4) στοιχεία.
  - Τη δεύτερη φορά: γίνονται αναζητήσεις για τους κωδικούς 0, 1 και 2. Διαγράφουμε τον κωδικό στη θέση 2 και εκεί έρχεται αυτός που ήταν στη θέση 4. Ο πίνακας μένει με 4 (0..3) στοιχεία.
  - Την τρίτη φορά: γίνονται αναζητήσεις για τους κωδικούς 0, 1, 2 και 3. Διαγράφουμε τον κωδικό στη θέση 3.
  - Την τέταρη φορά: γίνονται (επιτυχείς) αναζητήσεις για τους κωδικούς 0, 1 και 2.

Μπορούμε να αποφύγουμε τα παραπάνω αν ελέγχουμε τους κωδικούς των μαθημάτων μόλις τους διαβάσουμε και εισάγουμε στον πίνακα του αντικειμένου μόνον τους σωστούς. Έτσι το αντικείμενο εισάγεται στο μητρώο φοιτητών με την πρώτη. Στο παραπάνω παράδειγμα εισάγουμε μόνον τους κωδικούς 0, 1 και 3. Πάντως θα πρέπει να πάρεις υπόψη σου ότι αυτοί οι έλεγχοι θα (ξανα)γίνουν όταν καλέσουμε την *add1Student()*.

Ερχόμαστε τώρα στη διαχείριση των σφαλμάτων στη *δεύτερη περίπτωση*:

- Αν δεν βρεθεί ο αριθμός μητώου η εισαγωγή στο Μητρώο Φοιτητών γίνεται αφού δεν υπάρχουν άλλα προβλήματα. Αν βρεθεί ο αριθμός μητρώου δεν γίνεται η εισαγωγή αλλά έχουμε μια διαφορά από την πρώτη περίπτωση: πριν προχωρήσουμε στον επόμενο φοιτητή θα πρέπει να διαβάσουμε και να αγνοήσουμε τους κωδικούς των μαθημάτων.
- Αν εισαχθεί ο φοιτητής είσαγουμε και τις δηλώσεις μαθημάτων, καλώντας την *add1StudentInCourse()*, για το καθένα. Αν για κάποιο ακυρωθεί η εισαγωγή πιάνουμε την εξαίρεση και γράφουμε στο *log*. Εδώ οι έλεγχοι που θα γίνουν είναι μόνον αυτοί της *add1StudentInCourse()* και των συναρτήσεων που αυτή καλεί.

Όπως καταλαβαίνεις ο δεύτερος τρόπος είναι προτιμότερος· αυτόν θα υλοποιήσουμε.

Πριν από αυτό όμως θα πρέπει να δούμε τις εξαιρέσεις που μας ενδιαφέρουν, δηλαδή αυτές για τις οποίες θα γράψουμε καταχώριση στο *log*:

- Στην *add1StudentInCourse()* το πρώτο πράγμα που κάνουμε είναι να αναζητήσουμε την εγγραφή σε ένα αντικείμενο κλάσης *StudentInCourseCollection*. Για την αναζήτηση (*findNdx()*) θα πρέπει να δημιουργηθεί –από τον αριθμό μητρώου του φοιτητή και τον κωδικό του μαθήματος– ένα αντικείμενο κλάσης *StudentInCourse*. Αν η δομή του κωδικού μαθήματος δεν συμμορφώνεται με τους κανόνες (το μήκος να είναι 7)[10] *ο δημιουργός της StudentInCourse* θα ρίξει εξαίρεση *StudentInCourseXptn* με κωδικό *keyLen*.
- Αν δεν υπάρχει πρόβλημα με τη δομή του κωδικού μαθήματος γίνεται αναζήτηση στον πίνακα μαθημάτων· αν δεν βρεθεί μάθημα με τέτοιον κωδικό ρίχνεται εξαίρεση *StudentInCourseCollectionXptn* με κωδικό *unknownCrs* από την *add1StudentInCourse*.

Θα διαβάσουμε το αρχείο με τα στοιχεία των φοιτητών με την:

```
void readStudentData( string flNm,
                      StudentCollection& allStudents,
                      StudentInCourseCollection& allEnrollments,
                      CourseCollection& allCourses,
                      ofstream& log )
{
   ifstream tin( flNm.c_str() ); //"enrllmnt.txt" );
   if ( tin.fail() )
      throw ProgXptn( "readStudentData", ProgXptn::cannotOpen, flNm.c_str() );
   do {
      readAStudent( tin, allStudents, allEnrollments, allCourses, log );
   } while( !tin.eof() );
   tin.close();
} // readStudentData
```

που καλεί την παρακάτω συνάρτηση για να διαβάσει τα στοιχεία του κάθε φοιτητή και να ενημερώσει καταλλήλως τους πίνακες:

```
void readAStudent( istream& tin,
                   StudentCollection& allStudents,
                   StudentInCourseCollection& allEnrollments,
                   CourseCollection& allCourses,
                   ofstream& log )
{
   Student      aStudent;
   unsigned int sIdNum;

// Διάβασε τα στοιχεία ενός φοιτητή
```

[10] ή κάτι πιο πολύπλοκο, σαν αυτά που έχουμε στην άσκ. Prj03-1.

```
   aStudent.readPartFromText( tin );
   if ( !tin.eof() )
   {
      sIdNum = aStudent.getIdNum();
      if ( allStudents.find1Student(aStudent.getIdNum()) )
      { // υπάρχει στον πίνακα φοιτητών
      // Αγνόησε τη δήλωση
         ignoreStudentData( tin );
         log << " multiple entry for student with id num "
             << sIdNum << endl;
      }
      else
      {
      // Βάλε τον φοιτητή στον πίνακα φοιτητών
         allStudents.add1Student( aStudent );
         string str1;
         getline( tin, str1, '\n' );
         if ( !tin.eof() )
         {
            int noc( atoi(str1.c_str()) );
            for ( int k(0); k < noc && !tin.eof(); ++k )
            {
               getline( tin, str1, '\n' );
               try
               {
                  StudentInCourse aStdnInCrs( sIdNum, str1 );
                  allEnrollments.add1StudentInCourse( aStdnInCrs);
               }
               catch( StudentInCourseXptn& x )
               {
                  if ( x.errorCode == StudentInCourseXptn::keyLen )
                     log << " student with id num "
                         << aStudent.getIdNum()
                         << " asking course " << str1 << endl;
                  else
                     throw;
               } // catch( StudentInCourseXptn...
               catch( StudentInCourseCollectionXptn& x )
               {
                  if ( x.errorCode ==
                                  StudentInCourseCollectionXptn::unknownCrs )
                     log << " student with id num " << aStudent.getIdNum()
                         << " asking course " << str1 << endl;
                  else
                     throw;
               } // catch( StudentInCourseCollectionXptn...
            } // for
         } // if ( !tin.eof() )
         if ( !tin.eof() ) getline( tin, str1, '\n' );//blank line
      } // if ( allStudents.find1Student(aStudent.getIdNum()) )
   } // if ( !tin.eof() )
} // readAStudent
```

Η κάθε μια `catch` «μεταφράζει» σε C++ αυτά που είπαμε παραπάνω για τις εξαιρέσεις. Πρόσεξε ότι:

- Οι εξαιρέσεις άλλων τύπων –εκτός από τους δύο που μας ενδιαφέρουν– περνούν «ανενόχλητες».
- Ακόμη περνούν –αλλά με τη "`throw;`"– οι εξαιρέσεις που πιάνουμε αλλά έχουν κωδικό άλλον από αυτόν που μας ενδιαφέρει.

Prj04.11.2 Έλεγχος Δηλώσεων

Η πιο «φυσική» θέση της συνάρτησης για τον έλεγχο των δηλώσεων μαθημάτων (εβδομαδιαίου φόρτου) είναι ως μια μέθοδος της *StudentCollection*:

```
void StudentCollection::checkWH( ostream& log, int maxWH ) const
{
   if ( maxWH <= 0 )
      throw StudentCollectionXptn( "checkWH",
                                   StudentCollectionXptn::negWH, maxWH );
   for ( int k(0); k < scNOfStudents; ++k )
   {
      if ( scArr[k].getWH() > maxWH )
         log << "student with id num " << scArr[k].getIdNum() << ": "
             << scArr[k].getWH() << " hours/week" << endl;
   } // for
} // StudentCollection::checkWH
```

Αυτή καλείται από τη **main**, μετά την ανάγνωση των δεδομένων όλων των φοιτητών:

```
      unsigned int maxWH( 30 ); // μέγιστος επιτρεπόμενος φόρτος
                                // φοιτητή: 30 ώρες/εβδομάδα
      allStudents.checkWH( log, maxWH );
```

Prj04.11.3 Φύλαξη

Η φύλαξη των συλλογών (και του ευρετηρίου) γίνεται με κλήση της

```
void saveCollections( CourseCollection& allCourses,
                      StudentCollection& allStudents,
                      SIndexEntry* index,
                      StudentInCourseCollection& allEnrollments )
{
   ofstream bout( "Courses.dta", ios_base::binary );
   allCourses.save( bout );
   bout.close();
   bout.open( "students.dta", ios_base::binary );
   allStudents.save( bout, index );
   bout.close();
   bout.open( "students.ndx", ios_base::binary );
   for ( int k(0); k < allStudents.getNOfStudents(); ++k )
      bout.write( reinterpret_cast<const char*>(&index[k]),
                  sizeof(SIndexEntry) );
   bout.close();
   bout.open( "enrllmnt.dta", ios_base::binary );
   allEnrollments.save( bout );
   bout.close();
} // saveCollections
```

Αφού το ευρετήριο είναι πίνακας μπορεί να φυλαχθεί με πιο απλό τρόπο:

```
   bout.write( reinterpret_cast<const char*>(index),
               allStudents.getNOfStudents()*sizeof(IndexEntry) );
```

Prj04.11.4 ...Και το Πρόγραμμα

Το πρόγραμμα θα είναι ως εξής:

```
#include <string>
#include <fstream>
#include <new>
#include <iostream>

#include "MyTmpltLib.h"

using namespace std;

#include "Course.cpp"
#include "Student.cpp"
#include "StudentInCourse.cpp"
#include "StudentInCourseCollection.h"
#include "CourseCollection.h"
#include "SIndexEntry.h"
```

```
#include "StudentCollection.h"
#include "CourseCollection.cpp"
#include "StudentCollection.cpp"
#include "StudentInCourseCollection.cpp"

struct ProgXptn
{
   enum { allocFailed, cannotOpen };
   char functionName[100];
   int  errorCode;
   char errStrVal[100];
   ProgXptn( const char* fn, int ec, const char* sv="" )
      :  errorCode( ec )
   {  strncpy( functionName, fn, 99 );  functionName[99] = '\0';
      strncpy( errStrVal, sv, 99 );  errStrVal[99] = '\0';  }
}; // ProgXptn

void loadCourses( string flNm, CourseCollection& allCourses );
void readStudentData( string flNm,
                      StudentCollection& allStudents,
                      StudentInCourseCollection& allEnrollments,
                      CourseCollection& allCourses,
                      ofstream& log );
void saveCollections( CourseCollection& allCourses,
                      StudentCollection& allStudents,
                      IndexEntry* index,
                      StudentInCourseCollection& allEnrollments );

int main()
{
   try
   {
      CourseCollection allCourses;
      StudentCollection allStudents;
      StudentInCourseCollection allEnrollments;

      allEnrollments.setPAllStudents( &allStudents );
      allEnrollments.setPAllCourses( &allCourses );

      allCourses.setPAllEnrollments( &allEnrollments );
      allStudents.setPAllEnrollments( &allEnrollments );

      loadCourses( "Courses.dta", allCourses );

      ofstream log( "log.txt" );
      if ( log.fail() )
         throw ProgXptn( "main", ProgXptn::cannotOpen, "log.txt");

      readStudentData( "enrllmnt.txt", allStudents, allEnrollments,
                       allCourses, log );

      unsigned int maxWH( 30 ); // μέγιστος επιτρεπόμενος φόρτος
                                // φοιτητή: 30 ώρες/εβδομάδα
      allStudents.checkWH( log, maxWH );

      log.close();

      SIndexEntry* index;
      try
      {  index = new SIndexEntry[allStudents.getNOfStudents()];  }
      catch( bad_alloc )
      {  throw ProgXptn( "main", ProgXptn::allocFailed );  }

      saveCollections( allCourses, allStudents, index, allEnrollments );

      delete[] index;
   } // try
```

```
   catch( ProgXptn& x )  { /* . . . */ }
   catch( MyTmpltLibXptn& x )  { /* . . . */ }
   catch( CourseXptn& x )  { /* . . . */ }
   catch( StudentXptn& x )
   {
      switch ( x.errorCode )
      {
         case StudentXptn::allocFailed:
            cout << "Student " << x.objKey << ": no dynamic memory in "
                 << x.funcName << endl;
            break;
         case StudentXptn::negIdNum:
            cout << "Student " << x.objKey << ": illegal id num ("
                 << x.errIntVal << ") in " << x.funcName << endl;
            break;
         case StudentXptn::incomplete:
            cout << "incomplete data for Student " << x.objKey << " in "
                 << x.funcName << endl;
            break;
         case StudentXptn::fileNotOpen:
            cout << "file " << x.errStrVal << " not open in " << x.funcName
                 << endl;
            break;
         case StudentXptn::cannotWrite:
            cout << "cannot write to file " << x.errStrVal << " in "
                 << x.funcName << endl;
            break;
         case StudentXptn::cannotRead:
            cout << "cannot read from file " << x.errStrVal << " in "
                 << x.funcName << endl;
            break;
         default:
            cout << "unexpected StudentXptn for Student " << x.objKey
                 << " from " << x.funcName << endl;
      } // switch
   } // catch( StudentXptn
   catch( StudentInCourseXptn& x )  { /* . . . */ }
   catch( CourseCollectionXptn& x )  { /* . . . */ }
   catch( StudentCollectionXptn& x )  { /* . . . */ }
   catch( StudentInCourseCollectionXptn& x )  { /* . . . */ }
   catch( ... )
   {  cout << "unexpected exceptioon" << endl;  }
} // main
```

Να παρατηρήσουμε τα εξής:

- Αν αλλάξεις τη σειρά των οδηγιών "`include`" μπορεί να αντιμετωπίσεις προβλήματα.
- Η "`delete[] index`" είναι απαραίτητη; Σε κάθε περίπτωση, αφού εκεί τελειώνει το πρόγραμμα, η δυναμική μνήμη θα ελευθερωθεί! Σωστό! Τη βάλαμε μόνο και μόνο για να δεις ότι ενώ για την ανακύκλωση των δυναμικών πινάκων που είναι μέσα σε αντικείμενα θα δράσουν οι καταστροφείς για τον index θα πρέπει να ζητήσουμε ρητώς την ανακύκλωση.
- Όπως καταλαβαίνεις, δεν υπάρχει λόγος να παραθέσουμε το μακροσκελέστατο τμήμα διαχείρισης εξαιρέσεων –δηλαδή όλες τις `catch`– αφού οι μόνες που έχουν πραγματικό ενδιαφέρον είναι αυτές της *readAStudent* και όχι αυτές της **main**.
- Τέλος, πρόσεξε πόσο απλή είναι η **main** αφού όλη η πολυπλοκότητα έχει κρυφτεί μέσα στις μεθόδους.

Prj04.12 Το 2ο Πρόγραμμα – Εκμετάλλευση

Το δεύτερο πρόγραμμα είναι πολύ απλό:

```
#include <fstream>
```

```
#include <iostream>
#include <new>

#include "MyTmpltLib.h"

using namespace std;

#include "Course.cpp"
#include "Student.cpp"
#include "SIndexEntry.h"

struct ProgXptn
// ΟΠΩΣ ΣΤΟ 1ο ΠΡΟΓΡΑΜΜΑ

void loadIndex( string flNm, PIndexEntry& index, unsigned int& ndxSz );
void retrieve( string flNm, PIndexEntry index, int ndxSz );

int main()
{
   IndexEntry*  index;
   unsigned int ndxSz;
   try
   {
      loadIndex( "students.ndx", index, ndxSz );
      retrieve( "students.dta", index, ndxSz );
   } // try
   catch( ProgXptn& x )  { /* . . . */ }
   catch( MyTmpltLibXptn& x )  { /* . . . */ }
   catch( StudentXptn& x )  { /* . . . */ }
   catch( ... )
   {  cout << "unexpected exceptioon" << endl;  }
}
```

Στο `SIndexEntry.h` έχουμε τα εξής:

```
struct SIndexEntry
{
   unsigned int sIdNum;
   size_t       loc;
   explicit IndexEntry( int aIdNum=0, int aLoc=0 )
   {  sIdNum = aIdNum;  loc = aLoc;  }
}; // SIndexEntry

typedef SIndexEntry* PIndexEntry;

bool operator!=( const SIndexEntry& lhs, const SIndexEntry& rhs )
{  return ( lhs.sIdNum != rhs.sIdNum );  }

bool operator==( const SIndexEntry& lhs, const SIndexEntry& rhs )
{  return !(lhs != rhs);  }
```

Επιφορτώνουμε τον "`!=`" (και τον "`==`") και γράφουμε τον ερήμην δημιουργό με αρχικές τιμές διότι θα μας χρειαστούν για να χρησιμοποιήσουμε τη *linSearch()*. Δύο αντικείμενα *SIndexEntry* θεωρούνται ίσα αν και μόνον αν έχουν ίδια τιμή στο *sIdNum*. Φυσικά, θα παρατηρήσεις ότι ο δημιουργός χρειάζεται κάποιους ελέγχους. Σωστό! Γράψε τους (μαζί και μια κλάση εξαιρέσεων)!

Ας δούμε τώρα τις δύο συναρτήσεις. Η *loadIndex()* βασίζεται σε αυτά που μάθαμε στην §15.12.1:

```
void loadIndex( string flNm, PIndexEntry& index, unsigned int& ndxSz )
{
   ifstream bin( flNm.c_str(), ios_base::binary );
   if ( bin.fail() )
      throw ProgXptn( "loadIndex", ProgXptn::cannotOpen, flNm.c_str() );

   bin.seekg( 0, ios_base::end );
   unsigned int flSize( bin.tellg() );
```

```
   ndxSz = flSize/sizeof(IndexEntry);

   try {  index = new IndexEntry[ndxSz+1];  }
   catch( bad_alloc )
   {  throw ProgXptn( "loadIndex", ProgXptn::allocFailed );  }

   bin.seekg( 0 );
   for ( int k(0); k < ndxSz; ++k )
      bin.read( reinterpret_cast<char*>(&index[k]), sizeof(IndexEntry) );
   bin.close();
} // loadIndex
```

Με τις:

```
   bin.seekg( 0, ios_base::end );
   unsigned int flSize( bin.tellg() );
   ndxSz = flSize/sizeof(IndexEntry);
```

βρίσκουμε το μέγεθος του αρχείου και υπολογίζουμε το πλήθος των στοιχείων που περιέχονται σε αυτό.

Στη συνέχεια παίρνουμε την απαιτούμενη δυναμική μνήμη. Το "**+1**" μας δίνει μια θέση στον πίνακα για τον φρουρό που χρησιμοποιεί η *linSearch()*.

Τέλος, με τη **for** φορτώνουμε το περιεχόμενο του αρχείου στον πίνακα.

Η ανάκτηση και η επίδειξη των στοιχείων των φοιτητών γίνεται με τη:

```
void retrieve( string flNm, PIndexEntry index, int ndxSz )
{
   ifstream bin( flNm.c_str(), ios_base::binary );
   if ( bin.fail() )
      throw ProgXptn( "retrieve", ProgXptn::cannotOpen, flNm.c_str() );

   string line;
   cout << "Student Id Number: ";  getline( cin, line, '\n' );
   while ( line != "ΤΕΛΟΣ" )
   {
      int idNum( atoi(line.c_str()) );
      int ndx( linSearch(index, ndxSz, 0, ndxSz-1, IndexEntry(idNum)) );
      if ( ndx < 0 )
         cout << "unknown Student Id Number" << endl;
      else
      {
         Student oneStudent;
         bin.seekg( index[ndx].loc );
         oneStudent.load( bin );
         oneStudent.display( cout );
      }
      cout << "Student Id Number: ";  getline( cin, line, '\n' );
   } // while
   bin.close();
} // retrieve
```

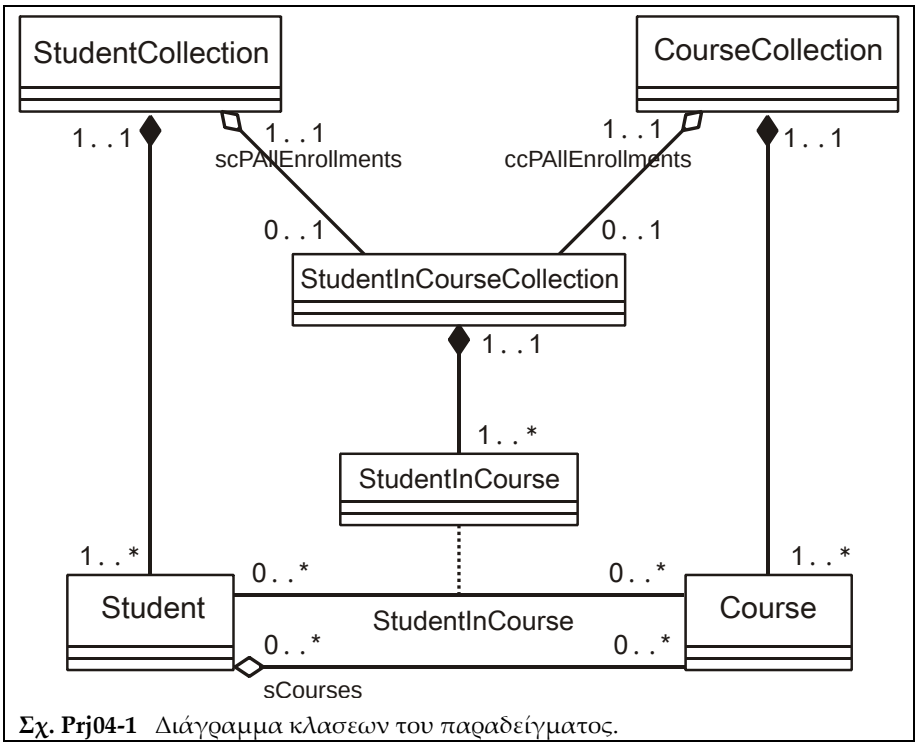
Η *retrieve()* τροφοδοτείται με τον index και το όνομα του αρχείου –με τα στοιχεία των φοιτητών– το οποίο και ανοίγει.

Στη συνέχεια ζητάει από τον χρήστη τον αριθμό μητρώου ενός φοιτητή και ψάχνει στον index να βρει στοιχείο με αυτόν. Αν το βρει –στη θέση *ndx*– φορτώνει τα στοιχεία του φοιτητή με τις:

```
         Student oneStudent;
         bin.seekg( index[ndx].loc );
         oneStudent.load( bin );
```

Στην §15.14.3 είχαμε υποσχεθεί ότι «*Αργότερα θα δούμε μια πολύ απλή μορφή ευρετηρίου.*» Εδώ πραγματοποιήσαμε αυτό που υποσχεθήκαμε. Αλλά... το παράδειγμά μας έχει πολύ περιορισμένη χρησιμότητα: Μόνο ανάκτηση πληροφοριών.

Δεν μπορούμε να κάνουμε και ενημέρωση; Μπορούμε, αρκεί να περιοριστούμε στα γενικά στοιχεία του φοιτητή και να μην αλλάξουμε το πλήθος μαθημάτων. Φυσικά δεν μπο-



Σχ. Prj04-1 Διάγραμμα κλασεων του παραδείγματος.

ρούμε να διαγράφουμε ή να προσθέτουμε νέους φοιτητές. Με τέτοιες ενημερώσεις χάνουμε τη δυνατότητα της σειριακής διαχείρισης του αρχείου. Για να τα έχουμε όλα χρειάζονται πέρα από το ευρετήριο και ειδικοί αλγόριθμοι που δεν έχουν σχέση με τους στόχους αυτού του βιβλίου.

Prj04.13 Για το Παράδειγμά μας

Με αυτό το παράδειγμα είχαμε την ευκαιρία να δούμε πώς χρησιμοποιούμε αυτά που μάθαμε μέχρι τώρα για κλάσεις και αντικείμενα. Στο Σχ. Prj04-1 βλέπεις και το σχετικό διάγραμμα κλάσεων.

Πόσο «πραγματικό» είναι το παράδειγμα; Όχι πολύ. Και αυτό δεν έχει σχέση μόνο με τα στοιχεία που θα πρέπει να κρατάει ένα αρχείο φοιτητών και λείπουν από το δικό μας. Ας δούμε τα άλλα προβλήματά του.

- Μια «πραγματική» εφαρμογή αυτού του είδους στήνεται με ένα *Σύστημα Διαχείρισης Βάσεων Στοιχείων* (Data Base Management Systems, DBMS) και όχι με απλά αρχεία. Πολλά από τα προβλήματα που αντιμετωπίσαμε[11] (και μερικά από αυτά που θα δούμε στη συνέχεια) αντιμετωπίζονται από το ΣΔΒΔ και η δουλειά του προγραμματιστή διευκολύνεται. Φυσικά, το ΣΔΒΔ κρατάει τους πίνακας στη ΒΔ, στον δίσκο, και όχι στη μνήμη.
- Το «σωστό» σχέδιο ήταν αυτό που είχαμε στο Project 3 και όχι το τωρινό.[12] Η αλήθεια είναι ότι στις «πραγματικές» εφαρμογές γίνονται τέτοιες «αποκανονικοποιήσεις» για να αυξήσουμε την ταχύτητα εκτέλεσης. Αλλά αυτές εισάγουν πλεονασμό και είδες πώς πληρώνεται ο πλεονασμός: οι μεγαλύτερες περιπλοκές σε αυτά που γράψαμε έχουν σχέση με τη συνεπή ενημέρωση του πίνακα δηλώσεων μαθημάτων και του πίνακα μαθημάτων του κάθε φοιτητή.

---

[11] Για παράδειγμα οι έλεγχοι ακεραιότητας οντότητας και αναφοράς.

[12] Στη γλώσσα του Σχεσιακού μοντέλου ΒΔ λέμε ότι ο πίνακας μαθημάτων μέσα στο αντικείμενο κλάσης *Student* παραβιάζει την *1η Κανονική Μορφή*.

- Θα μπορούσαμε να έχουμε τον πλεονασμό μόνο στην κύρια μνήμη αλλά όχι στα αρχεία: Φυλάγουμε στο αρχείο φοιτητών για κάθε φοιτητή μόνον επώνυμο, όνομα και αριθμό μητρώου. Κατα τη φόρτωση των στοιχείων, τα υπόλοιπα συμπληρώνονται όταν φορτώνουμε το αρχείο δηλώσεων μαθημάτων. Έτσι δεν έχουμε εγγραφές μεταβλητού μήκους και δεν χρειαζόμαστε ευρετήριο (τουλάχιστον για τον λόγο αυτόν).
- Και κάτι ακόμη: Τα στοιχεία του φοιτητή επώνυμο, όνομα και αριθμός μητρώου είναι σταθερά για όλη τη διάρκεια των σπουδών του. Τα άλλα, πλήθος και κωδικοί μαθημάτων και εβδομαδιαίος φόρτος, έχουν σχέση με ένα συγκεκριμένο ακαδημαϊκό εξάμηνο και επαναλαμβάνονται. Για κάθε φοιτητή λοιπόν θα πρέπει να έχουμε ένα αντικείμενο με τα:

```
   unsigned int sIdNum;        // αριθμός μητρώου
   char         sSurname[sNameSz];
   char         sFirstname[sNameSz];
// άλλα στοιχεία που παραλείψαμε
```

και πολλά –ένα για κάθε ακαδημαϊκό εξάμηνο– με τα

```
   unsigned int       sIdNum;        // αριθμός μητρώου
   char               sSemester[10]; // ακαδημαϊκό εξάμηνο
   unsigned int       sWH;           // ώρες ανά εβδομάδα
   unsigned int       sNoOfCourses;  // αριθμός μαθημάτων που
                                     // δήλωσε
   Course::CourseKey* sCourses;
```

- Παρομοίως, για κάθε μάθημα θα πρέπει να έχουμε ένα αντικείμενο με τα «σταθερά» στοιχεία:

```
   CourseKey    cCode;            // κωδικός μαθήματος
   char         cTitle[cTitleSz]; // τίτλος μαθήματος
   unsigned int cFSem;            // τυπικό εξάμηνο
   bool         cCompuls;         // υποχρεωτικό ή επιλογής
   char         cSector;          // τομέας
   char         cCateg[cCategSz]; // κατηγορία
   unsigned int cWH;              // ώρες ανά εβδομάδα
   unsigned int cUnits;           // διδακτικές μονάδες
   CourseKey    cPrereq;          // προαπαιτούμενο
```

και ένα με τα στοιχεία που αλλάζουν κάθε εξάμηνο:

```
   CourseKey    cCode;            // κωδικός μαθήματος
   unsigned int cNoOfStudents;    // αριθ. Φοιτητών
```

(και ακόμη τους αριθμούς μητρώου σπουδαστών που το παρακολουθούν, τα στοιχεία του διδάσκοντα κλπ.)

- Σε ένα τέτοιο σύστημα προγραμμάτων και δεδομένων, σε πραγματικές συνθήκες, θα πρέπει να έχουν ταυτόχρονη πρόσβαση πολλοί χρήστες –για παράδειγμα υπάλληλοι της γραμματείας του τμήματος και πιθανότατα οι ίδιοι οι ενδιαφερόμενοι φοιτητές– χωρίς όμως να καταστρέφουν ο ένας τη δουλειά του άλλου. Οι κίνδυνοι προέρχονται από τον τρόπο λειτουργίας των πολυχρηστικών ΛΣ που δίνουν εκ περιτροπής χρόνο εξυπηρέτησης σε όλους τους χρήστες αλλά με πολύ μεγάλη –για τον άνθρωπο– ταχύτητα ώστε να μην ενοχλείται ο χρήστης. Σκέψου λοιπόν την εξής περίπτωση:
  - Υπάλληλος της γραμματείας ζητάει φύλαξη των πινάκων (με τη συνάρτηση *saveCollections()*) του πρώτου προγράμματος· φυλάγεται ο πίνακας μαθημάτων, φυλάγεται το μητρώο των φοιτητών και…
  - Στο σημείο εκείνο το ΛΣ διακόπτει την εκτέλεση του προγράμματος και εξυπηρετεί την απαίτηση ενός φοιτητή που αλλάζει τις δηλώσεις μαθημάτων.
  - Όταν θα επιστρέψει για να συνεχίσει τη φύλαξη θα φυλάξει τον πίνακα δηλώσεων που όμως δεν είναι συνεπής με αυτούς που είχαν φυλαχθεί πριν τη διακοπή.

  Ένα ΣΔΒΔ, με τη δυνατότητα **επεξεργασίας συναλλαγών** (transaction processing) που σου παρέχει, σου επιτρέπει να απαλλαγείς με σχετικώς απλό τρόπο από τέτοιες επι-

πλοκές. Αλλιώς θα πρέπει να καταφύγεις στη χρήση εργαλείων που δίνει το ΛΣ (σημαφόροι, κλειδώματα κ.ά.) που κάνουν το πρόγραμμά σου πολύ πιο πολύπλοκο.