project

7

Φοιτητές και Μαθήματα με STL

Περιεχόμενα:

Εισαγωγικό Σημείωμα:

Στόχος αυτού του Project είναι να ξαναδούμε το γνωστό μας πρόβλημα χρησιμοποιώντας αυτά που μάθαμε για την STL. Θα αλλάξουμε τη λύση που είδαμε στο Project 4 με δύο τρόπους:

- Στην πρώτη περίπτωση θα αλλάξουμε όλους τους δυναμικούς πίνακες σε *vector*.
- Στη δεύτερη περίπτωση θα επιλέγουμε το περιέχον που μας φαίνεται πιο κατάλληλο.

Και στις δύο περιπτώσεις, θα κάνουμε τις μετατροπές υπακούοντας στον εξής περιορισμό: *Δεν θα πρέπει να αλλάζει ο τρόπος που καλούνται οι μέθοδοι που είναι "*`public`*" ώστε* να μην χρειάζεται να αλλάξουμε τα προγράμματα που χρησιμοποιούν τις κλάσεις μας. Αυτό σημαίνει ότι οι μέθοδοι θα πρέπει:

- Να έχουν τις ίδιες προδιαγραφές (προϋπόθεση-απαίτηση).
- Να έχουν την ίδια επικεφαλίδα.

Για τις μεθόδους που είναι "`private`" είμαστε πιο ελαστικοί.

Όπως θα δεις στη συνέχεια, θα παραβιάσουμε τον παραπάνω περιορισμό μόνον στην περίπτωση μιας συγκεκριμένης λειτουργίας: στην εξαγωγή ολόκληρου πίνακα (όπως, για παράδειγμα, ο ccArr της *CourseCollection*).

Prj07.1 Το Πρόγραμμα με STL I: *vector*

Θα αλλάξουμε τη λύση του Project 4 αντικαθιστώντας με το κατάλληλο στιγμιότυπο του "`vector`" κάθε πίνακα που έχουμε στη λύση. Πέρα από αυτό μπορεί να αλλάζουμε και μερικά από αυτά που έχουμε γράψει με κάτι «καλό» από την STL.

Prj07.1.1 Κλάση *Course*

Η κλάση *Course* δεν έχει πίνακες και έτσι δεν αλλάζει.

Prj07.1.2 Κλάση *CourseCollection*

Εδώ η βασική αλλαγή είναι στη δήλωση του ccArr:

```
   vector<Course> ccArr;
```

Τα *ccIncr*, *ccNOfCourses*, *ccReserved* μας είναι άχρηστα. Η *getNOfCourses()* γίνεται:

```
   size_t getNOfCourses() const {  return ccArr.size();  }
```

Με τη *getArr()* τι θα κάνουμε; Μια λύση είναι:

```
   const Course* getArr() const {  return &ccArr[0];  }
```

Θα το ξανασυζητήσουμε…

Πέρα από τα παραπάνω, η πρώτη απλούστευση της κλάσης φαίνεται στον δημιουργό:

```
   CourseCollection() { };
```

και στον καταστροφέα που δεν τον χρειάζόμαστε! Εμείς όμως, κατά τη συνήθειά μας θα γράψουμε:

```
   ~CourseCollection() { };
```

Θα ξεκινήσουμε τις μετατροπές από την (εσωτερική) συνάρτηση-μέλος *findNdx()* που χρησιμοποιείται παντού. Αντί για τη *linSearch()* (για να γίνουμε «πιο STL») θα χρησιμοποιήσουμε την (*std*::)*find()*:

```
vector<Course>::iterator CourseCollection::findNdx( string code )
{
   vector<Course>::iterator ndx;
   if ( code.length() != Course::cCodeSz-1 )
      ndx = ccArr.end();
   else
      ndx = find( ccArr.begin(), ccArr.end(), Course(code) );
   return ndx;
} // CourseCollection::findNdx
```

Η *findNdx()* επέστρεφε τιμή τύπου `int` που ήταν δείκτης στοιχείου του ccArr (δηλαδή `ccArr[ndx]`). Τώρα επιστρέφει έναν προσεγγιστή προς στοιχείο του πίνακα· έτσι:

όπου βάζαμε "`ccArr[ndx]`" θα βάζουμε "`*ndx`" και

όπου βάζαμε "`ccArr[ndx].`*X*" θα βάζουμε "`ndx->`*X*"

Ακόμη, οι πράξεις "`++ccNOfCourses`" και "`--ccNOfCourses`" δεν έχουν νόημα και θα πρέπει να αφαιρεθούν.

Παρατήρηση:▶

Για σύνδεση με τα προηγούμενα κρατούμε το όνομα "*findNdx*" που όμως είναι συντομογραφία του «*find index*» (βρες δείκτη). Τώρα –που η μέθοδος δεν επιστρέφει δείκτη αλλά προσεγγιστή– θα έπρεπε να αλλάξουμε το όνομα σε κάτι σαν "*findIt*" για το «*find iterator*» (βρες προσεγγιστή).◀

Ξεκινούμε με τη:

```
   bool find1Course( const string& code ) const
   {  return ( findNdx(code) != ccArr.end() );  };
```

Συνεχίζουμε με τη *delete1Course()*:

```
void CourseCollection::delete1Course( string code )
{
   vector<Course>::iterator ndx( findNdx(code) );
   if ( ndx != ccArr.end() )  // υπάρχει
   {
      vector<Course>::iterator k;
      k = find_if( ccArr.begin(), ccArr.end(),
                   bind2nd(EqPrereq(),code) );
      if ( k != ccArr.end() )
         throw CourseCollectionXptn( "delete1Course",
                                     CourseCollectionXptn::prereqRef,
                                      code.c_str() );
      int enrStdnt( ndx->getNoOfStudents() );
      if ( enrStdnt > 0 ) // υπάρχουν εγγραφές φοιτητών
         throw CourseCollectionXptn( "delete1Course",
                                     CourseCollectionXptn::enrollRef,
                                      code.c_str(), enrStdnt );
      erase1Course( ndx );
   }
} // CourseCollection::delete1Course
```

Εδώ, πέρα από τις αλλαγές που έχουν σχέση με την αλλαγή τύπου του *ndx*, έχουμε και κάτι άλλο: ενσωματώσαμε αυτό που είδαμε στο παράδειγμα της §26.1.4. Εδώ βέβαια το *k* είναι τύπου `vector<Course>::iterator` και αντί για `ccArr`, `ccArr+ccNOfCourses` έχουμε `ccArr.begin()`, `ccArr.end()` αντιστοίχως. Η *EqPrereq* είναι ίδια:

```
struct EqPrereq :  binary_function< Course, string, bool >
{
   bool operator()( const Course& x, const string& y ) const
   {  return ( strcmp(x.getPrereq(), y.c_str()) == 0 );  }
}; // EqPrereq
```

Πρόσεξε ότι πρέπει να αλλάξει η

```
void CourseCollection::erase1Course( vector<Course>::iterator ndx )
{
  *ndx = ccArr.back();
  ccArr.pop_back();
} // CourseCollection::erase1Course
```

Και προχωρούμε στην *add1Course()*:

```
void CourseCollection::add1Course( const Course& aCourse )
{
   if ( strlen(aCourse.getCode()) != Course::cCodeSz-1 )
      throw CourseCollectionXptn( "add1Course",
                                  CourseCollectionXptn::entity );
   vector<Course>::iterator ndx( findNdx(aCourse.getCode()) );
   if ( ndx == ccArr.end() ) // δεν βρέθηκε το κλειδί
   {
      if ( strcmp(aCourse.getPrereq(), "") != 0 )  // υπάρχει
      {                                            // προαπαιτούμενο
         vector<Course>::iterator ndx(
                                    findNdx(aCourse.getPrereq()) );
         if ( ndx == ccArr.end() ) // δεν βρέθηκε το κλειδί του
                                   // προαπαιτούμενου
            throw CourseCollectionXptn( "add1Course",
                                    CourseCollectionXptn::prereqRef,
                                        aCourse.getPrereq() );
      }
   // δεν υπάρχει προαπαιτούμενο ή υπάρχει και βρέθηκε στον πίνακα
      insert1Course( aCourse );
      ndx = findNdx( aCourse.getCode() );
      ndx->clearStudents();
```

```
   }
} // CourseCollection::add1Course
```

όπου:

```
void CourseCollection::insert1Course( const Course& aCourse )
{
   try {  ccArr.push_back( aCourse );  }
   catch ( bad_alloc& )
   {  throw CourseCollectionXptn( "insert1Course",
                              CourseCollectionXptn::allocFailed );  }
} // CourseCollection::insert1Course
```

Εδώ, όπως περιμέναμε, η απλούστευση είναι εντυπωσιακή. Βέβαια, η πιθανότητα να «ξεμείνουμε» από μνήμη υπάρχει και παίρνουμε τα μέτρα μας.

Η τελευταία μέθοδος «ενός στοιχείου» είναι η:

```
const Course& CourseCollection::get1Course( string code ) const
{
   vector<Course>::const_iterator ndx( findNdx(code) );
   if ( ndx == ccArr.end() )
      throw CourseCollectionXptn( "get1Course",
                                  CourseCollectionXptn::notFound,
                                  code.c_str() );
   return *ndx;
} // CourseCollection::get1Course
```

Πρόσεξε το "`const_iterator ndx`": αυτό είναι απαραίτητο συμπλήρωμα των "`const`" της επικεφαλίδας.

Πριν δούμε τις μεθόδους φύλαξης-φόρτωσης ας δούμε τη

```
void CourseCollection::swap( CourseCollection& rhs )
{
   ccArr.swap( rhs.ccArr );
} // CourseCollection::swap
```

που όπως βλέπεις απλουστεύθηκε.

Η φύλαξη γίνεται με τη:

```
void CourseCollection::save( ofstream& bout ) const
{
   if ( bout.fail() )
      throw  CourseCollectionXptn( "save",
                                   CourseCollectionXptn::fileNotOpen );
   size_t  ccNOfCourses( ccArr.size() );
   bout.write( reinterpret_cast<const char*>(&ccNOfCourses),
               sizeof(ccNOfCourses) );
   for ( vector<Course>::const_iterator it(ccArr.begin());
         it != ccArr.end(); ++it )
      it->save( bout );
   if ( bout.fail() )
      throw  CourseCollectionXptn( "save",
                                   CourseCollectionXptn::cannotWrite );
} // CourseCollection::save
```

και η φόρτωση με τη:

```
void CourseCollection::load( ifstream& bin )
{
   CourseCollection tmp;
   size_t n;  //ccNOfCourses

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

Η *load()* μένει όπως ήταν. Αυτό οφείλεται στο ότι δηλώσαμε την *tmp* τύπου *CourseCollection*. Αν επιλέξουμε "`vector<Course> tmp`" τότε αλλάζουμε:

τη "`tmp.insert1Course(oneCourse)`" σε "`tmp.push_back(oneCourse)`"

και τη "`swap(tmp)`" σε "`ccArr.swap(tmp)`"

και η φόρτωση γίνεται κάπως πιο γρήγορα.

Η *display()* δεν αλλάζει παρά μόνο στη συνθήκη της **for**. Θα πρέπει να γίνει "`k < ccArr.size()`".

Οι *add1Student()* και *delete1Student()* –ενδιάμεσες προς τις μεθόδους της *Course* με το ίδιο όνομα– αλλάζουν όπως είπαμε στην αρχή της παραγράφου:

```
void CourseCollection::add1Student( string code )
{
   vector<Course>::iterator ndx( findNdx(code) );
   if ( ndx == ccArr.end() ) // δεν υπάρχει τέτοιο μάθημα
      throw CourseCollectionXptn( "add1Student",
                                  CourseCollectionXptn::notFound,
                                  code.c_str() );
   ndx->add1Student();
} // CourseCollection::add1Student
```

και

```
void CourseCollection::delete1Student( string code )
{
   vector<Course>::iterator ndx( findNdx(code) );
   if ( ndx == ccArr.end() ) // δεν υπάρχει τέτοιο μάθημα
      throw CourseCollectionXptn( "delete1Student",
                                  CourseCollectionXptn::notFound,
                                  code.c_str() );
   ndx->delete1Student();
} // CourseCollection::delete1Student
```

Prj07.1.2.1 Σχετικώς με τη *getArr()*

Υποσχεθήκαμε να ξανασυζητήσουμε για τη μέθοδο *CourseCollection*::*getArr()* και αυτό θα κάνουμε αλλά πριν προχωρήσεις καλό είναι να γυρίσεις στην §20.7.5 και να διαβάσεις το σημείο "**5. Μέθοδος** *getAllStops()*".

Εκεί είχαμε καταλήξει ότι το καλύτερο που είχαμε να κάνουμε για να «βγάλουμε» έναν πίνακα από το αντικείμενο ήταν να επιστρέψουμε δυναμικό πίνακα, αντίγραφο του εσωτερικού. Βέβαια επισημάναμε τον κίνδυνο «διαρροής» μνήμης και στηρίζαμε τις ελπίδες μας για την αποφυγή της στις καλές συνήθειες των πεπειραμένων προγραμματιστών.[1]

Τώρα θα δούμε μια πιο σίγουρη λύση, με παράδειγμα εφαρμογής στον ccArr:

```
void CourseCollection::getArr( vector<Course>& crsArr ) const
{
   try  {  crsArr = ccArr;  }
   catch( bad_alloc& )
   {  throw CourseCollectionXptn( "getArr",
                                  CourseCollectionXptn::allocFailed );  }
} // CourseCollection::getArr
```

Αυτή η λύση είναι στην πραγματικότητα μια μορφή RAII. Όταν δώσεις:

```
   CourseCollection allCourses;
// . . .
   vector<Course> courseTbl;
```

[1] Που, όμως, όταν ρίχνονται εξαιρέσεις, μπορεί να μην είναι αρκετές.

```
// . . .
   allCourses.getArr( courseTbl );
```

η δυναμική μνήμη που παίρνουμε κρύβεται μέσα στην *courseTbl*. Όταν τελειώσει η ζωή της ο καταστροφέας της θα ανακυκλώσεί αυτομάτως και τη μνήμη.

Φυσικά, θα υπενθυμίσουμε ότι και αυτή η «καλύτερη λύση» κοστίζει σε υπολογιστικό χρόνο αφού έχουμε αντιγραφή πίνακα.

Με αυτόν τον τρόπο θα βγάζουμε όλους τους πίνακες στη συνέχεια. Στο σημείο αυτό θα αλλάξουν οι διεπαφές όλων των κλάσεων.

Prj07.1.3 Κλάση *Student*

Η *Student* αλλάζει αφού θα αντικαταστήσουμε τα «εργαλεία υλοποίησης» του δυναμικού πίνακα μαθημάτων:

```
   enum { sIncr = 3 };
   size_t             sNoOfCourses;  // αριθμός μαθημάτων που δήλωσε
   Course::CourseKey* sCourses;
   size_t             sReserved;
```

με τη

```
   vector<Course::CourseKey> sCourses;
```

Ύστερα από αυτό θα έχουμε:

```
   unsigned int getNoOfCourses() const {  return sCourses.size();  }
   void clearCourses() {  sCourses.clear();  sWH = 0;  }
```

Φυσικά, οι πράξεις "`++sNoOfCourses`" και "`--sNoOfCourses`" και εδώ δεν έχουν νόημα και θα πρέπει να αφαιρεθούν.

Γράφουμε τη *getCourses()* όπως κάναμε στην *CourseCollection* με τη *getArr()*:

```
void Student::getCourses( vector<Course::CourseKey>& crsTbl ) const
{
   try {  crsTbl = sCourses;  }
   catch( bad_alloc& )
   {  throw StudentXptn( sIdNum, "getCourses",
                         StudentXptn::allocFailed );  }
} // Student::getCourses
```

Αρχίζουμε με τους δημιουργούς: Ο ερήμην δημιουργός γίνεται:

```
Student::Student( int aIdNum )
   :  sWH( 0 )
{
   if ( aIdNum < 0 )
      throw StudentXptn( 0, "Student", StudentXptn::negIdNum,
                         aIdNum );
   sIdNum = aIdNum;
   sSurname[0] = '\0';
   sFirstname[0] = '\0';
}; // Student()
```

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 = rhs.sCourses;  }
   catch( bad_alloc& )
   {
      throw StudentXptn( sIdNum, "Student",
                         StudentXptn::allocFailed );
   }
} // Student( const Student& rhs )
```

Τώρα πρόσεξε: *δημουργός αντιγραφής δεν χρειάζεται κατ' αρχήν* αφού το περίγραμμα *vector* έχει (σωστό) δημιουργό αντιγραφής. Και γιατί το βάλαμε; Για να πιάνουμε και να αλλάζουμε τη *bad_alloc*.

Παρατήρηση:▶

Και για τον ερήμην δημιουργό δεν χρειάζεται να κάνουμε το ίδιο; Κατ' αρχήν: ναι!◀

Θα συνεχίσουμε με τη *findNdx()*· την παίρνουμε από την *CourseCollection*::*findNdx()* αν βάλουμε:

- Τύπο προσεγγιστή

  "`vector<Course::CourseKey>::iterator`"

  (αντί για "`vector<Course>::iterator`").
- Όνομα πίνακα "`sCourses`" (αντί για "`ccArr`").
- Αντικείμενο αναζήτησης "`Course::CourseKey(code)`" (αντί για "`Course(code)`").

```
vector<Course::CourseKey>::iterator Student::findNdx( const string& code )
{
   vector<Course::CourseKey>::iterator ndx;
   if ( code.length() != Course::cCodeSz-1 )
      ndx = sCourses.end();
   else
      ndx = find( sCourses.begin(), sCourses.end(),
                  Course::CourseKey(code) );
   return ndx;
} // Student::findNdx
```

Και μετά από αυτήν οι τρεις μέθοδοι «ενός στοιχείου»:

```
   bool find1Course( const string& code ) const
   {  return ( findNdx(code) != sCourses.end() );  };

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

void Student::delete1Course( const Course& oneCourse )
{
   vector<Course::CourseKey>::iterator
                                 ndx( findNdx(oneCourse.getCode()) );
   if ( ndx != sCourses.end() )  // υπάρχει
   {
      erase1Course( ndx );
      sWH -= oneCourse.getWH();
   }
} // Student::delete1Course
```

όπου:

```
void Student::insert1Course( const Course::CourseKey& aCode )
{
   try {  sCourses.push_back( aCode );  }
   catch( bad_alloc& )
   {
      throw StudentXptn( sIdNum, "insert1Course",
                         StudentXptn::allocFailed );
   }
} // Student::insert1Course

void Student::erase1Course( vector<Course::CourseKey>::iterator ndx)
{
   *ndx = sCourses.back();
   sCourses.pop_back();
```

```
} // Student::erase1Course
```

Από τις υπόλοιπες μεθόδους αλλάζουν και οι:

```
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 );

   sCourses.swap( rhs.sCourses );
} // Student::swap

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) );
   size_t noOfCourses( sCourses.size() );
   bout.write( reinterpret_cast<const char*>(&noOfCourses),
               sizeof(noOfCourses) );// αριθμός μαθημάτων που δήλωσε
   for ( int k(0); k < noOfCourses; ++k )
      bout.write( sCourses[k].s, Course::cCodeSz );
   if ( bout.fail() )
      throw StudentXptn( sIdNum, "save", StudentXptn::cannotWrite );
} // Student::save

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) );
      size_t noOfCourses;              // αριθ. μαθημ. που δήλωσε
      bin.read( reinterpret_cast<char*>(&noOfCourses), sizeof(noOfCourses) );
      for ( int k(0); k < noOfCourses; ++k )
      {
         char aKey[Course::cCodeSz];
         bin.read( aKey, Course::cCodeSz );
         tmp.sCourses.push_back( Course::CourseKey(aKey) );
      }
      if ( bin.fail() )
         throw StudentXptn( sIdNum, "load",
                            StudentXptn::cannotRead );
      swap( tmp );
   }
} // Student::load
```

Prj07.1.4 Κλάση *StudentCollection*

Εδώ, όπως κάναμε στις προηγούμενες κλάσεις, θα αντικαταστήσουμε τις

```
   enum { scIncr = 30 };
```

```
   Student*                scArr;
   size_t                  scNOfStudents;
   size_t                  scReserved;
```

με τη

```
   vector<Student>            scArr;
```

Και όπως μάθαμε, θα πρέπει να αλλάξουμε τις:

```
   size_t getNOfStudents() const {  return scArr.size();  }

void StudentCollection::getArr( vector<Student>& stdntTbl ) const
{
   try {  stdntTbl = scArr;  }
   catch( bad_alloc& )
   {  throw StudentCollectionXptn( "getArr",
                            StudentCollectionXptn::allocFailed );  }
} // StudentCollection::getArr
```

Ο δημιουργός και ο καταστροφέας απλουστεύονται:

```
StudentCollection::StudentCollection()
{  scPAllEnrollments = 0;  }
   ~StudentCollection() { };
```

Η συνέχεια είναι γνωστή:

```
vector<Student>::iterator StudentCollection::findNdx( int aIdNum )
{
   vector<Student>::iterator ndx;
   if ( aIdNum <= 0 )
      ndx = scArr.end();
   else
      ndx = find( scArr.begin(), scArr.end(), Student(aIdNum) );
   return ndx;
} // StudentCollection::findNdx

   bool find1Student( int aIdNum ) const
   {  return ( findNdx(aIdNum) != scArr.end() );  };

const Student& StudentCollection::get1Student( int aIdNum ) const
{
   vector<Student>::const_iterator ndx( findNdx(aIdNum) );
   if ( ndx == scArr.end() )
      throw StudentCollectionXptn( "get1Student",
                                   StudentCollectionXptn::notFound,
                                   aIdNum );
   return *ndx;
} // StudentCollection::get1Student

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

Στην *add1Student()* πρόσεξε την αλλαγή στον χειρισμό του *aStCourses* λόγω της αλλαγής της *Student*::*getCourses()*.

```
void StudentCollection::delete1Student( int aIdNum )
{
   vector<Student>::iterator ndx( findNdx(aIdNum) );
   if ( ndx != scArr.end() )  // υπάρχει
   {
      if ( ndx->getNoOfCourses() > 0 )
         throw StudentCollectionXptn( "delete1Student",
                                      StudentCollectionXptn::enrollRef,
                                      aIdNum );
      erase1Student( ndx );
   }
} // StudentCollection::delete1Student

void StudentCollection::insert1Student( const Student& aStudent )
{
   try {  scArr.push_back( aStudent );  }
   catch( MyTmpltLibXptn& )
   {
      throw StudentCollectionXptn( "insert1Student",
                                   StudentCollectionXptn::allocFailed );
   }
} // StudentCollection::insert1Student

void StudentCollection::erase1Student(vector<Student>::iterator ndx)
{
   *ndx = scArr.back();
   scArr.pop_back();
} // StudentCollection::erase1Student
```

Εδώ έχουμε επιπλέον:

```
void StudentCollection::add1Course( int aIdNum, const Course& aCourse )
{
   vector<Student>::iterator ndx( findNdx(aIdNum) );
   if ( ndx == scArr.end() )
      throw StudentCollectionXptn( "add1Course",
                                   StudentCollectionXptn::notFound,
                                   aIdNum );
   ndx->add1Course( aCourse );
} // StudentCollection::add1Course

void StudentCollection::delete1Course( int aIdNum, const Course& aCourse )
{
   vector<Student>::iterator ndx( findNdx(aIdNum) );
   if ( ndx == scArr.end() )
      throw StudentCollectionXptn( "delete1Course",
                                   StudentCollectionXptn::notFound,
                                   aIdNum );
   ndx->delete1Course( aCourse );
} // StudentCollection::delete1Course
```

Τέλος, έχουμε τις:

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

void StudentCollection::save( ofstream& bout, SIndexEntry* index )
{
   if ( bout.fail() )
      throw StudentCollectionXptn( "save",
                                   StudentCollectionXptn::fileNotOpen );
   size_t nOfStudents( scArr.size() );
   bout.write( reinterpret_cast<const char*>(&nOfStudents),
               sizeof(nOfStudents) );
```

```
   for ( int k(0); k < nOfStudents; ++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

void StudentCollection::load( ifstream& bin )
{
   StudentCollection tmp;
   size_t n;
   bin.read( reinterpret_cast<char*>(&n), sizeof(size_t) );
   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
```

Ένα ερώτημα για τη δεύτερη παράμετρο της *save()*: Δεν θα αλλάξουμε τον πίνακα του ευρετηρίου; Θα τον αλλάξουμε· είπαμε όμως να μην αλλάξουμε τις διεπαφές των κλάσεων. Θα τα πούμε στη συνέχεια…

Prj07.1.5 Κλάση *StudentInCourse*

Η κλάση *StudentInCourse* δεν χρειάζεται τροποποίηση.

Prj07.1.6 Κλάση *StudentInCourseCollection*

Εδώ, όπως καταλαβαίνεις, θα δηλώσουμε τον δυναμικό πίνακα ως:

```
   vector<StudentInCourse> siccArr;
```

Αυτό έχει τις εξής άμεσες επιπτώσεις:

```
StudentInCourseCollection::StudentInCourseCollection()
{
   siccPAllStudents = 0;
   siccPAllCourses = 0;
} // StudentInCourseCollection::StudentInCourseCollection

   ~StudentInCourseCollection()  {  };

   size_t getNOfStudentInCourses() const  {  return siccArr.size();  }

void StudentInCourseCollection::getArr( vector<StudentInCourse>& sicTbl) const
{
   try {  sicTbl = siccArr;  }
   catch( bad_alloc& )
   {  throw StudentInCourseCollectionXptn( "getArr",
                    StudentInCourseCollectionXptn::allocFailed );  }
} // StudentInCourseCollection::getArr
```

Και –κατά τη συνήθειά μας– συνεχίζουμε με τη:

```
vector<StudentInCourse>::iterator
                StudentInCourseCollection::findNdx( int aIdNum, string code )
{
   vector<StudentInCourse>::iterator ndx;
   if ( aIdNum <= 0 || code.length() != Course::cCodeSz-1 )
      ndx = siccArr.end();
   else
      ndx = find( siccArr.begin(), siccArr.end(),
                  StudentInCourse(aIdNum,code) );
   return ndx;
} // StudentInCourseCollection::findNdx
```

Μετά από αυτήν:

```
   bool find1StudentInCourse( int aIdNum, string code ) const
   {  return ( findNdx(aIdNum, code) != siccArr.end() );  }

const StudentInCourse&
          StudentInCourseCollection::get1StudentInCourse( int aIdNum,
                                                          string code ) const
{
   vector<StudentInCourse>::const_iterator
                                        ndx( findNdx(aIdNum, code) );
   if ( ndx == siccArr.end() ) // δεν υπάρχει στοιχείο
      throw StudentInCourseCollectionXptn( "get1StudentInCourse",
                           StudentInCourseCollectionXptn::notFound,
                           StudentInCourse::SICKey(aIdNum, code) );
   return *ndx;
} // StudentInCourseCollection::get1StudentInCourse
```

Στην *add1StudentInCourse()* το μόνο που αλλάζει είναι η συνθήκη της πρώτης **if**:

```
void StudentInCourseCollection::add1StudentInCourse(
                                        const StudentInCourse& aStdnInCrs )
{
   if ( findNdx( aStdnInCrs.getSIdNum(),     // δεν υπάρχει φοιτητής
                 aStdnInCrs.getCCode()) == siccArr.end() )
   {
   // . . .
      insert1StudentInCourse( aStdnInCrs );
   }
} // StudentInCourseCollection::add1StudentInCourse
```

Η *insert1StudentInCourse()* απλουστεύεται:

```
void StudentInCourseCollection::insert1StudentInCourse(
                                        const StudentInCourse& aStdnInCrs )
{
   try {  siccArr.push_back( aStdnInCrs );  }
   catch( MyTmpltLibXptn& )
   {
      throw StudentInCourseCollectionXptn( "insert1StudentInCourse",
                                  StudentInCourseCollectionXptn::allocFailed );
   }
} // StudentInCourseCollection::insert1StudentInCourse
```

Για τις διαγραφές έχουμε:

```
void StudentInCourseCollection::delete1StudentInCourse( int aIdNum,
                                                        string code )
{
   vector<StudentInCourse>::iterator ndx( findNdx(aIdNum, code) );
   if ( ndx != siccArr.end() ) // 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
```

όπου:

```
void StudentInCourseCollection::erase1StudentInCourse(
                                      vector<StudentInCourse>::iterator ndx )
{
   *ndx = siccArr.back();
   siccArr.pop_back();
} // StudentInCourseCollection::erase1StudentInCourse
```

Και η *swapArr()* απλουστεύεται:

```
void StudentInCourseCollection::swapArr( StudentInCourseCollection& rhs )
{
   siccArr.swap( rhs.siccArr );
} // StudentInCourseCollection::swapArr
```

ενώ οι *save()* και *load()* γίνονται:

```
void StudentInCourseCollection::save( ofstream& bout ) const
{
   if ( bout.fail() )
      throw  StudentInCourseCollectionXptn( "save",
                                 StudentInCourseCollectionXptn::fileNotOpen );
   size_t nOfStudentInCourse( siccArr.size() );
   bout.write( reinterpret_cast<const char*>(&nOfStudentInCourse),
               sizeof(nOfStudentInCourse) );
   for ( int k(0); k < nOfStudentInCourse; ++k )
      siccArr[k].save( bout );
   if ( bout.fail() )
      throw  StudentInCourseCollectionXptn( "save",
                        StudentInCourseCollectionXptn::cannotWrite );
} // StudentInCourseCollection::save

void StudentInCourseCollection::load( ifstream& bin )
{
   StudentInCourseCollection tmp;
   size_t n;
   bin.read( reinterpret_cast<char*>(&n), sizeof(size_t) );
   if ( !bin.eof() )
   {
      for ( int k(0); k < n && !bin.fail(); ++k )
      {
         StudentInCourse oneStudentInCourse;
         oneStudentInCourse.load( bin );
         tmp.insert1StudentInCourse( oneStudentInCourse );
      }
      if ( bin.fail() )
         throw  StudentInCourseCollectionXptn( "load",
                                    StudentInCourseCollectionXptn::cannotRead );
      swapArr( tmp );
   }
} // StudentInCourseCollection::load
```

Prj07.1.7 Ο Πίνακας-Ευρετήριο

Είπαμε πιο πριν ότι δεν θα αλλάξουμε την επικεφαλίδα της *StudentCollection*::*save()* αλλά είπαμε στην αρχή ότι θα αλλάξουμε όλους τους πίνακες σε αντικείμενα στιγμιοτύπων του *vector*. Αυτά τα δύο είναι συμβατά; Ναι! Στη `main()` δίνουμε:

```
      vector<SIndexEntry> sIndex;
      try
      {  sIndex.reserve( allStudents.getNOfStudents() );  }
      catch( bad_alloc )
      {  throw ProgXptn( "main", ProgXptn::allocFailed );  }
```

και στη συνέχεια:

```
saveCollections( allCourses, allStudents, &sIndex[0], allEnrollments );
```

Η *saveCollections()* θα καλέσει:

```
allStudents.save( bout, sIndex );
```

Πρόσεξε το εξής: Προϋπόθεση της *StudentCollection*::*save()* είναι η ύπαρξη των στοιχείων του πίνακα sIndex. Αυτό μας το εξασφαλίζει η (επιτυχής) εκτέλεση της *sIndex*.*reserve()*. Αν δεν τη βάλουμε θα πρέπει να αλλάξουμε τη *save()* και να κάνουμε τις καταχωρίσεις με *push_back()*.

Prj07.2 Το Πρόγραμμα με STL II

Τώρα θα (ξανα)αλλάξουμε τη λύση του Project 4 αντικαθιστώντας κάθε πίνακα που έχουμε στη λύση με το καταλληλότερο –κατά περίπτωση– περιέχον.

Prj07.2.1 Επιλογές

Ξεκινούμε εξετάζοντας τους πίνακες έναν προς έναν για να δούμε με ποιο περιέχον θα αντικαταστήσουμε τον καθένα. Τι εξετάζουμε; Τη χρήση τους στα δύο προγράμματα του Project. Πάντως το δεύτερο πρόγραμμα το βλέπουμε γενικότερα, όπως θα έπρεπε να είναι, για τις ανάγκες της γραμματείας ενός τμήματος σπουδών.

Prj07.2.1.1 ccArr της *CourseCollection*

Σε αντικείμενο κλάσης *CourseCollection* έχουμε τον πίνακα μαθημάτων που υπάρχουν στο πρόγραμμα του τμήματος.

Ο πίνακας μαθημάτων είναι ένα *σύνολο* αντικειμένων τύπου *Course*. Ένα τέτοιο αντικείμενο είναι αρκετά μεγάλο (περί τα 100 B) ενώ το κλειδί είναι μικρό (8 B). Κατ' αρχήν θα πρέπει να παρασταθεί με στιγμιότυπο του *map*.

Ας δούμε όμως και τη χρήση του πίνακα:

- Στο 1ο Πρόγραμμα: ο πίνακας μαθημάτων, αφού φορτωθεί, χρησιμοποιείται μόνο για αναζητήσεις.
- Γενικότερα: Το πρόγραμμα ενός τμήματος αλλάζει μια φορά κάθε 3-4 χρόνια. Επομένως, η συνήθης χρήση του πίνακα μαθημάτων είναι η αναζήτηση πληροφοριών για τα μαθήματα που περιέχει.

Επειδή δεν έχουμε συχνή ενημέρωση του πίνακα θα εξετάσουμε και αυτό που λέμε στο τέλος της §26.4: ταξινομημένο *vector*. Δηλαδή:

- Στο 1ο Πρόγραμμα: ο πίνακας μαθημάτων, φορτώνεται σε ένα `vector <Course>`, ταξινομείται, στη συνέχεια οι αναζητήσεις γίνονται με τη *lower_bound()* και φυλάγεται ταξινομημένος.
- Στα άλλα προγράμματα ο πίνακας φορτώνεται ταξινομημένος και οι αναζητήσεις γίνονται με τη *lower_bound()*.

Στην περίπτωση αυτή θα έχουμε τα εξής προβλήματα:

- Όπως λέγαμε στην §26.4 «*είναι δική σου υποχρέωση να διασφαλίζεις τη μοναδικότητα των μελών του συνόλου.*» Αυτό το κάναμε ήδη στη λύση που έχουμε δώσει (§Prj07.1.2).
- Για να εκμεταλλευτούμε τα πλεονεκτήματα της επιλογής αυτής θα πρέπει να κάνουμε το εξής: Να κρατούμε τον πίνακα ταξινομημένο από την αρχή και να κάνουμε τις εισαγωγές με *insert()* στη σωστή θέση (και όχι με *push_back()*).

Παρατήρηση:▶
Γιατί να μην κάνουμε τις εισαγωγές στον πίνακα όπως την κάναμε στην πρώτη μετατροπή (με *push_back()*), στη συνέχεια να ταξινομήσουμε τον πίνακα και από εκεί και πέρα να ψάχνουμε με *lower_bound()*; Κάτι τέτοιο απαιτεί να έχουμε «διπλές» μεθόδους *findNdx()*, *find1Course()* και *get1Course()*. Αυτό είναι «εγγύηση» για δύσκολα λάθη στη συνέχεια! Ας το αφήσουμε καλύτερα…◀

Πιο πολύ για επίδειξη, θα επιλέξουμε τη λύση με το «ταξινομημένο *vector*». Αν θέλεις να δοκιμάσεις την απεικόνιση από τους κωδικούς μαθημάτων προς αντικείμενα *Course* καλύτερα να μιμηθείς αυτά που θα κάνουμε στη συνέχεια για τη *StudentCollection* και όχι αυτά που κάνουμε στο πρώτο παράδειγμα της §26.3.2.

Prj07.2.1.2 sCourses της *Student*

Εδώ έχουμε ένα πολύ μικρό *σύνολο* κωδικών μαθημάτων (κλειδιών).

Το σύνολο δημιουργείται στην αρχή του εξαμήνου και μπορεί να διορθωθεί μια φορά.[2] Χαρακτηριστική περίπτωση "*set*", αλλά μόνο για λόγους αρχής. Λόγω του μεγέθους του συνόλου δεν έχουμε κέρδος στις αναζητήσεις. Θα δεις ότι έχουμε κάτι μικρά κέρδη στις εισαγωγές και διαγραφές.

Prj07.2.1.3 scArr της *StudentCollection*

Αυτή η συλλογή έχει μερικές εκατοντάδες μέχρι μερικές χιλιάδες μέλη. Ενημερώσεις; Πολύ συχνές!

- Εισαγωγές με τις εγγραφές νέων φοιτητών.
- Διαγραφές με τις αποφοιτήσεις.
- Εισαγωγές με μετεγγραφές φοιτητών *από* άλλα τμήματα.
- Διαγραφές με μετεγγραφές φοιτητών *προς* άλλα τμήματα.
- Διαγραφές λόγω μη ανανέωσης εγγραφής.

Κάθε αντικείμενο της συλλογής είναι μοναδικό (τα στοιχεία ενός φοιτητή)· έχουμε δηλαδή ένα *σύνολο*.

Το μέγεθος του κλειδιού (αριθμός μητρώου του φοιτητή) είναι λιγότερο από 10% του μεγέθους του αντικειμενου. Εξ άλλου υπάρχουν και ενημερώσεις (π.χ. αλλαγές στο sCourses κάποιου φοιτητή) που δεν είναι εισαγωγές/διαγραφές.

Θα κάνουμε την υλοποίηση με "*map*", απεικόνιση των αριθμών μητρώου (κλειδιά) σε αντικείμενα *Student*.

Prj07.2.1.4 siccArr της *StudentInCourseCollection*

Εδώ έχουμε συλλογή με περίπου πενταπλάσιο αριθμό μελών από το προηγούμενο, αφού κάθε φοιτητής δηλώνει κατά μέσον όρο πέντε μαθήματα.

Κάθε αντικείμενο της συλλογής –δήλωση ενός φοιτητή για συμμετοχή σε ένα μάθημα για το τρέχον εξάμηνο– είναι μοναδικό· δηλαδή έχουμε και εδώ ένα *σύνολο*.

Ενημερώσεις; Αρκετές:

- Δηλώσεις φοιτητών για συμμετοχή στα μαθήματα στην αρχή του εξαμήνου.
- Διορθώσεις (διαγραφές-εγγραφές) των δηλώσεων.
- Ακυρώσεις των δηλώσεων όσων φοιτητών μετεγγράφονται *προς* άλλα τμήματα.

Στο τέλος του εξαμήνου έχουμε και άλλη ενημέρωση: εισαγωγή βαθμού.

Τι περιέχον θα επιλέξουμε; Πολλά τα βάλε-βγάλε άρα ούτε σκέψη για "*vector*".

---

[2] Δηλαδή νομίμως μια φορά. Διότι με τις «κλάψες» στη γραμματεία μπορεί να έχουμε και άλλες διορθώσεις.

Το κλειδί (αριθμός μητρώου, κωδικός μαθήματος) είναι –σε μέγεθος– το 75% του αντικειμένου. Μήπως το απλούστερο που έχουμε να επιλέξουμε είναι το "*set*"; Αν κάνουμε αυτήν την επιλογή, πώς θα βάζουμε τους βαθμούς; Κατ' ανάγκη με τις πράξεις: «βγάλε-διάγραψε-διόρθωσε-ξαναβάλε». Αν επιλέξουμε "*map*" θα έχουμε τη δυνατότητα για ενημέρωση επί τόπου.

Θα επιλέξουμε "*map*".

Prj07.2.1.5 Πίνακας–Ευρετήριο

Πόσο μας απλούστευσε το πρόγραμμα η αλλαγή που κάναμε στην §Prj07.1.7; Καθόλου! Ο απλός δυναμικός πίνακας είναι εξ ίσου απλός (αν όχι απλούστερος) με το στιγμιότυπο του *vector*. Φυσικά αυτό δεν είναι περίεργο:

- παίρνουμε όλη τη μνήμη που χρειαζόμαστε με μια `new[]` και την ανακυκλώνουμε με μια `delete[]`,
- συμπληρώνουμε τα στοιχεία με πολύ απλό τρόπο,
- έχουμε τη δυνατότητα να κάνουμε τη φύλαξη του με μια *write()*.

Κατ' αρχή λοιπόν δεν θα αλλάξουμε τη διαχείριση αυτού του πίνακα. Αλλά ας δούμε τι κάνουμε (ξανά και ξανά) στο δεύτερο πρόγραμμα:

- παίρνουμε έναν αριθμό μητρώου φοιτητή,
- βρίσκουμε από το ευρετήριο τη θέση του αντίστοιχου αντικειμένου στο αρχείο με τα στοιχεία των φοιτητών,
- το φορτώνουμε και
- το δείχνουμε.

Είπαμε όμως ότι η συλλογή στοιχείων φοιτητών «*έχει μερικές εκατοντάδες μέχρι μερικές χιλιάδες μέλη.*» Εδώ η ταχύτητα αναζήτησης έχει νόημα.[3] Τι να κάνουμε; Δεν θα αλλάξουμε περιέχον αλλά θα ταξινομήσουμε τον πίνακα (κατ' αύξοντα αριθμό μητρώου) ώστε να υπάρχει δυνατότητα δυαδικής αναζήτησης από τα προγράμματα εφαρμογής που τον χρησιμοποιούν.

Prj07.2.2 Κλάση *Course*

Όπως θα δεις παρακάτω, η *CourseCollection* θέτει απαίτηση για επιφόρτωση του τελεστή "`<`" για την *Course*: [4]

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

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

Prj07.2.3 Κλάση *CourseCollection*

Όπως στην §Prj07.1.2 έτσι και εδώ δηλώνουμε:

```
   vector<Course> ccArr;
```

[3] Θα πεις: και αυτό «κατ' αρχήν» διότι το βήμα «*παίρνουμε έναν αριθμό μητρώου φοιτητή*» εκτελείται με ανθρώπινες ταχύτητες· και θα έχεις δίκιο.

[4] Για λόγους συνέπειας με τους κανόνες μας, βάζουμε ακόμη:

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

και έχουμε:

```
   CourseCollection() { };
   ~CourseCollection() { };
   size_t getNOfCourses() const {  return ccArr.size();  }
   const Course* getArr() const {  return &ccArr[0];  }
```

Τη *getArr()* θα την αλλάξουμε όπως στην §Prj07.1.2.1.

```
void CourseCollection::getArr( vector<Course>& crsArr ) const
// ΟΠΩΣ ΠΑΡΑΠΑΝΩ
```

Να δούμε όμως τι γίνεται με τη *findNdx()*. Τώρα δεν θα καλέσουμε τη *find()* αλλά τη *lower_bound()*:

```
vector<Course>::iterator CourseCollection::findNdx( const string& code )
{
   vector<Course>::iterator ndx;
   if ( code.length() != Course::cCodeSz-1 )
      ndx = ccArr.end();
   else
      ndx = lower_bound( ccArr.begin(), ccArr.end(), Course(code) );
   return ndx;
} // CourseCollection::findNdx
```

Πρόσεξε τώρα: η *lower_bound()* θα μας επιστρέψει προσεγγιστή προς το αντικείμενο που ψάχνουμε (αν υπάρχει) ή προς τη θέση που πρέπει να εισαχθεί (αν δεν υπάρχει). Επομένως η *find1Course()* θα γίνει έτσι:

```
bool CourseCollection::find1Course( const string& code ) const
{
   vector<Course>::iterator ndx( findNdx(code) );
   return ( (ndx != ccArr.end()) && (*ndx == Course(code)) );
} // CourseCollection::find1Course
```

Η συνθήκη στη `return` είναι η διατύπωση της απόφανσης «*το βρήκα*». Όπως λέγαμε στην §5.6, αν η "`ndx != ccArr.end()`" υπολογιστεί `false` η συνάρτηση επιστρέφει `false` χωρίς να κάνει την (απαγορευμένη) αποπαραπομπή του *ndx*. Η διατύπωση της «*δεν το βρήκα*» είναι η άρνηση της παράστασης της `return` που ισοδυναμεί με:

"`ndx == ccArr.end() || (*ndx != Course(code))`"

Και εδώ δεν κινδυνεύουμε από απαγορευμένη αποπαραπομπή.

Για να χρησιμοποιήσουμε αυτή τη μορφή της *lower_bound()* χρειάζεται να έχουμε ορίσει τον "`<`" για την *Course*.

Η *CourseCollection*::*delete1Course()* είναι ίδια με αυτήν της §Prj07.1.2 με μια μόνο διαφορά: στη συνθήκη της **if** που ελέγχει αν υπάρχει το προς διαγραφή αντικείμενο που τώρα γίνεται:

```
   if ( ndx != ccArr.end() && (*ndx == Course(code)) )  // υπάρχει
```

Ίδια είναι και η *EqPrereq*. Πιο σοβαρή είναι η διαφορά στην *erase1Course()* που τώρα γίνεται:

```
void CourseCollection::erase1Course( vector<Course>::iterator ndx )
{
   ccArr.erase( ndx );
} // CourseCollection::erase1Course
```

Γράφουμε ολόκληρη την *add1Course()* στη νέα της μορφή:

```
void CourseCollection::add1Course( const Course& aCourse )
{
   if ( strlen(aCourse.getCode()) != Course::cCodeSz-1 )
      throw CourseCollectionXptn( "add1Course",
                                  CourseCollectionXptn::entity );
   vector<Course>::iterator ndx( findNdx(aCourse.getCode()) );
   if ( ndx==ccArr.end() || (*ndx!=aCourse) )
   {                                          // δεν βρέθηκε το κλειδί
      if ( strcmp(aCourse.getPrereq(), "") != 0 )  // υπάρχει
      {                                           // προαπαιτούμενο
```

```
         vector<Course>::iterator ndx(findNdx(aCourse.getPrereq()));
         if ( ndx == ccArr.end() ) // δεν βρέθηκε το κλειδί του
                                   // προαπαιτούμενου
            throw CourseCollectionXptn( "add1Course",
                                        CourseCollectionXptn::prereqRef,
                                            aCourse.getPrereq() );
      }
   // δεν υπάρχει προαπαιτούμενο ή υπάρχει και βρέθηκε στον πίνακα
      ndx = insert1Course( ndx, aCourse );
      ndx->clearStudents();
   }
} // CourseCollection::add1Course
```

Ποιες είναι οι διαφορές;

- Η συνθήκη που ελέγχει την τιμή του *ndx*.
- Αλλάξαμε την *insert1Course()* ώστε να επιστρέφει προσεγγιστή προς το αντικείμενο που εισάχθηκε στον πίνακα όπως τον επιστρέφει η *ccArr.insert()*. Έτσι, δεν είναι απαραίτητο να αναζητήσουμε το αντικείμενο που μόλις βάλαμε στον πίνακα.

```
vector<Course>::iterator
               CourseCollection::insert1Course( vector<Course>::iterator ndx,
                                                const Course& aCourse )
{
   try {  return ccArr.insert( ndx, aCourse );  }
   catch ( bad_alloc& )
   {  throw CourseCollectionXptn( "insert1Course",
                             CourseCollectionXptn::allocFailed );  }
} // CourseCollection::insert1Course
```

Η *get1Course()* γίνεται:

```
const Course& CourseCollection::get1Course( string code ) const
{
   vector<Course>::const_iterator ndx( findNdx(code) );
   if ( ndx == ccArr.end() || (*ndx != Course(code)) )
      throw CourseCollectionXptn( "get1Course",
                                  CourseCollectionXptn::notFound,
                                  code.c_str() );
   return *ndx;
} // CourseCollection::get1Course
```

Δηλαδή, το μόνο που άλλαξε είναι ο έλεγχος του *ndx* στην **if**.

Το ίδιο ισχύει και για τις επόμενες:

```
void CourseCollection::add1Student( string code )
{
   vector<Course>::iterator ndx( findNdx(code) );
   if ( ndx==ccArr.end() || (*ndx!=Course(code)) )
      throw CourseCollectionXptn( "add1Student",
                                  CourseCollectionXptn::notFound,
                                  code.c_str() );
   ndx->add1Student();
} // CourseCollection::add1Student

void CourseCollection::delete1Student( string code )
{
   vector<Course>::iterator ndx( findNdx(code) );
   if ( ndx==ccArr.end() || (*ndx!=Course(code)) )
      throw CourseCollectionXptn( "delete1Student",
                                  CourseCollectionXptn::notFound,
                                  code.c_str() );
   ndx->delete1Student();
} // CourseCollection::delete1Student
```

Η *swap()* γίνεται όπως στην §Prj07.1.2· το ίδιο και η *save()*.

Η *load()* πρέπει να αλλάξει λίγο: αφού αλλάξαμε την *insert1Course()* αντί "`tmp.insert1Course(oneCourse)`" θα πρέπει να δώσουμε "`tmp.add1Course( oneCourse )`".

Prj07.2.4 Κλάση *Student*

Για το σύνολο κωδικών μαθημάτων δηλώνουμε:

```
   set<Course::CourseKey> sCourses;
```

και ύστερα από αυτό έχουμε (όπως στην §Prj07.1.3):

```
   unsigned int getNoOfCourses() const {  return sCourses.size();  }
   void clearCourses() {  sCourses.clear();  sWH = 0;  }
```

Η *getCourses()* αλλάζει λίγο:

```
void Student::getCourses( vector<Course::CourseKey>& crsTbl ) const
{
   try
   {
      crsTbl.clear();
      for ( set<Course::CourseKey>::const_iterator it(sCourses.begin());
            it != sCourses.end(); ++it )
         crsTbl.push_back( *it );
   }
   catch( bad_alloc& )
   {  throw StudentXptn( sIdNum, "getCourses", StudentXptn::allocFailed );  }
} // Student::getCourses
```

Τώρα δεν μπορούμε να περάσουμε τους κωδικούς με μια εκχώρηση· πρέπει να διασχίσουμε όλο το σύνολο και να τους περνούμε έναν προς έναν.

Οι δύο δημιουργοί είναι σαν αυτούς που γράψαμε στην §Prj07.1.3, τουλάχιστον οπτικώς. Αλλά

- Στον ερήμην δημιουργό δεν φαίνεται ότι δημιουργείται το (κενό σύνολο) *sCourses*.
- Στον δημιουργο αντιγραφής η "`sCourses = rhs.sCourses`" είναι εκχώρηση με σύνολα.

Να επισημάνουμε ότι ούτε εδώ χρειάζεται δημιουργός αντιγραφής αλλά τον αφήνουμε για να πιάνει και να αλλάζει την (πιθανή) *bad_alloc*. Για τον ίδιο λόγο (και μόνον) αφήνουμε και τον `Student::operator=()`. Όσο για τον καταστροφέα, αυτός δεν έχει τι να κάνει:

```
   ~Student() { };
```

Και προχωρούμε στη *findNdx()*. Μπορούμε να την πάρουμε από αυτήν της §Prj07.1.3 αλλάζοντας το "`vector<Course::CourseKey>::iterator`" σε "`set<Course::CourseKey>::iterator`"; Κατ' αρχήν ναι. Αλλά είναι προτιμότερο να κάνουμε άλλη μια αλλαγή:

```
set<Course::CourseKey>::iterator Student::findNdx( const string& code )
{
   set<Course::CourseKey>::iterator ndx;
   if ( code.length() != Course::cCodeSz-1 )
      ndx = sCourses.end();
   else
      ndx = sCourses.find( Course::CourseKey(code) );
   return ndx;
} // Student::findNdx
```

Η *sCourses.find()* ψάχνει σε χρόνο $O(log(sCourses.size()))$ αντί για το $O(sCourses.size())$ της αναζήτησης της *std::find()*.

Η *find1Course()* γίνεται, όπως και στην §Prj07.1.3:

```
   bool find1Course( const string& code )
   {  return ( findNdx(code) != sCourses.end() );  };
```

Θα αλλάξουμε όμως τις *add1Course()* και *delete1Course()* αχρηστεύοντας τις (εσωτερικές) *insert1Course()* και *erase1Course()*:

```
typedef pair< set<Course::CourseKey>::iterator, bool > InsRetType;

void Student::add1Course( const Course& oneCourse )
{
   try
   {
      InsRetType rv(sCourses.insert(Course::CourseKey(oneCourse.getCode())) );
```

```
      if ( rv.second )  sWH += oneCourse.getWH();
   }
   catch( bad_alloc& )
   {
      throw StudentXptn( sIdNum, "insert1Course", StudentXptn::allocFailed );
   }
} // Student::add1Course
```

Να δούμε τι κάναμε εδώ: Στην §26.3.1 είπαμε ότι η μέθοδος *insert()* του *set* (1η μορφή) παίρνει ως όρισμα το κλειδί που θέλουμε να εισαγάγουμε –στην περίπτωσή μας: `Course::CourseKey(oneCourse.getCode())`– και επιστρέφει μια τιμή *pair* (στην περίπτωσή μας: *rv*):

- Το πρώτο μέλος της (*rv*.*first*) είναι προσεγγιστής του τύπου συνόλου που δουλεύουμε (`set<Course::CourseKey>::iterator`) προς το κλειδί όπως βρίσκεται μέσα στο σύνολο.
- Το δεύτερο μέλος της (*rv*.*second*) είναι τιμή `bool` που έχει τιμή `true` αν έγινε η εισαγωγή ή`false` αν δεν έγινε εισαγωγή διότι η τιμή υπήρχε.

Αν το *rv*.*second* είναι `true` προσθέτουμε στις εβδομαδιαίες ώρες του φοιτητή τις εβδομαδιαίες ώρες του μαθήματος.

Παρομοίως, στη *delete1Course()*, αξιοποιούμε την τιμή που επιστρέφει η *erase()* του *set*:

```
void Student::delete1Course( const Course& oneCourse )
{
   size_t erased( sCourses.erase(Course::CourseKey(oneCourse.getCode())) );
   if ( erased > 0 )  sWH -= oneCourse.getWH();
} // Student::delete1Course
```

Φυλάγουμε την τιμή που επιστρέφει η *sCourses*.*erase()* στη μεταβλητή *erased*.

- Αν το κλειδί δεν υπήρχε στο σύνολο τότε η τιμή της γίνεται "`0`".
- Αλλιώς, αν υπήρχε, γίνεται "`1`" και επειδή ισχύει η "`erased > 0`" αφαιρούμε από τις εβδομαδιαίες ώρες του φοιτητή τις εβδομαδιαίες ώρες του μαθήματος.

Η *swap()* παραμένει όπως έγινε στην §Prj07.1.3 αλλά η εντολή "`sCourses.swap(rhs.sCourses)`" ανταλλάσσει τιμές συνόλων.

Στη *save()* υπάρχει μια υποχρεωτική αλλαγή: Η **for** δεν μπορεί να ελέγχεται πια με τον δείκτη του στοιχείου πίνακα· θα ελέγχεται από προσεγγιστή που διατρέχει ολόκληρο σύνολο:

```
   for( set<Course::CourseKey>::const_iterator it(sCourses.begin());
        it != sCourses.end(); ++it )
      bout.write( it->s, Course::cCodeSz );
```

Παρόμοια αλλαγή θα πρέπει να γίνει και στη *display()*.

Στη *load()*, βρίσκουμε τη διαφορά όταν φτάνουμε στην ανάγνωση των κωδικών μαθημάτων:

```
      for ( int k(0); k < noOfCourses; ++k )
      {
         char aKey[Course::cCodeSz];
         bin.read( aKey, Course::cCodeSz );
         tmp.sCourses.insert( Course::CourseKey(aKey) );
      }
```

Λόγω της αλλαγής περιέχοντος, η κλήση της *tmp*.*sCourses*.*insert()* αντικαθιστά την κλήση της *tmp*.*sCourses*.*push_back()*.

Prj07.2.5 Κλάση *StudentCollection*

Είπαμε στην §Prj07.1.1.4 ότι «*Θα κάνουμε την υλοποίηση με "map", απεικόνιση των αριθμών μητρώου (κλειδιά) σε αντικείμενα Student.*» Θα δηλώσουμε:

```
   map< unsigned int, Student > scArr;
```

Ο τύπος κλειδιών είναι "`unsigned int`" και ο τύπος τιμών "`Student`". Θα πεις: αφού το κλειδί είναι ο αριθμός μητρώου και αφού αυτό υπάρχει μέσα στο *Student* (μέλος *sIdNum*)

δεν έχουμε πλεονασμό (που, όπως είναι γνωστό, είναι επικίνδυνος); Έχουμε! Αλλά, θα είμαστε προσεκτικοί στους χειρισμούς μας. Θα κάνουμε εισαγωγές στοιχείων στο scArr με την:

"`scArr[aStudent.getIdNum()] = aStudent`"

ή με την:

"`scArr.insert( make_pair(aStudent.getIdNum(),aStudent) )`"

Σε σχέση με την §Prj07.1.4, δεν έχουμε αλλαγή στη

```
size_t getNOfStudents() const {  return scArr.size();  }
```

αλλά έχουμε αλλαγή στη

```
void StudentCollection::getArr( vector<Student>& stdntTbl ) const
{
   try
   {
      for ( map<unsigned int,Student>::const_iterator it(scArr.begin());
            it != scArr.end();  ++it )
         stdntTbl.push_back( it->second );
   }
   catch( bad_alloc& )
   {  throw StudentCollectionXptn( "getArr",
                                   StudentCollectionXptn::allocFailed );  }
} // StudentCollection::getArr
```

Ο *it* στη **for** δείχνει ένα ζεύγος (κλειδί, αντικείμενο). Το "`it->second`" είναι το αντικείμενο με τα στοιχεία φοιτητή. Σύγκρινε τη *getArr()* με τη *getCourses()* της *Student*.

Δημιουργός και καταστροφέας δεν διαφέρουν οπτικώς από αυτούς που γράψαμε στην §Prj07.1.4.

Πάμε τώρα στη

```
map< unsigned int, Student >::iterator StudentCollection::findNdx( int aIdNum)
{
   map<unsigned int,Student>::iterator ndx;
   if ( aIdNum <= 0 )
      ndx = scArr.end();
   else
      ndx = scArr.find( aIdNum );
   return ndx;
} // StudentCollection::findNdx
```

Τι δείχνει ο *ndx* και ο προσεγγιστής που επιστρέφει η *findNdx()*; Ένα ζεύγος τύπου `pair<unsigned int,Student>`.

Η *find1Student()* γίνεται όπως στην §Prj07.1.4. Πρόσεξε όμως πώς γίνεται η

```
const Student& StudentCollection::get1Student( int aIdNum ) const
{
   map<unsigned int,Student>::const_iterator ndx( findNdx(aIdNum) );
   if ( ndx == scArr.end() )
      throw StudentCollectionXptn( "get1Student",
                                   StudentCollectionXptn::notFound, aIdNum );
   return ndx->second;
} // StudentCollection::get1Student
```

Σε σχέση με αυτήν που γράψαμε στην §Prj07.1.4, δεν άλλαξε μόνον ο τύπος του *ndx* αλλά και η επιστρεφόμενη τιμή που τώρα είναι "`ndx->second`".

Ο τύπος του *ndx* αλλάζει και στην *add1Student()* αλλά εδώ αλλάζουμε και κάτι άλλο: αχρηστεύουμε την *insert1Student()*:

```
void StudentCollection::add1Student( const Student& aStudent )
{
   map<unsigned int,Student>::iterator ndx( findNdx(aStudent.getIdNum()) );
   if ( ndx == scArr.end() ) // δεν βρέθηκε το κλειδί
   {
      if ( scPAllEnrollments == 0 )
         throw StudentCollectionXptn( "add1Student",
                                      StudentCollectionXptn::noEnroll );
```

```
      scArr.insert( make_pair(aStudent.getIdNum(),aStudent) );
      vector<Course::CourseKey> aStCourses;
      aStudent.getCourses( aStCourses );
      for ( int k(0); k < aStudent.getNoOfCourses(); ++k )
      {
         scPAllEnrollments->add1StudentInCourse(
                     StudentInCourse(aStudent.getIdNum(), aStCourses[k].s) );
      }
   }
} // StudentCollection::add1Student
```

Παρομοίως, στη *delete1Student()* αχρηστεύουμε την *erase1Student()*:

```
void StudentCollection::delete1Student( int aIdNum )
{
   map<unsigned int,Student>::iterator ndx( findNdx(aIdNum) );
   if ( ndx != scArr.end() )  // υπάρχει
   {
      if ( (ndx->second).getNoOfCourses() > 0 )
         throw StudentCollectionXptn( "delete1Student",
                                      StudentCollectionXptn::enrollRef,
                                      aIdNum );
      scArr.erase( ndx );
   }
} // StudentCollection::delete1Student
```

Οι ενδιάμεσες μέθοδοι εισαγωγής/διαγραφής μαθημάτων σε έναν φοιτητή γίνονται:

```
void StudentCollection::add1Course( int aIdNum, const Course& aCourse )
{
   map<unsigned int,Student>::iterator ndx( findNdx(aIdNum) );
   if ( ndx == scArr.end() )
      throw StudentCollectionXptn( "add1Course",
                                   StudentCollectionXptn::notFound, aIdNum );
   (ndx->second).add1Course( aCourse );
} // StudentCollection::add1Course

void StudentCollection::delete1Course( int aIdNum, const Course& aCourse )
{
   map<unsigned int,Student>::iterator ndx( findNdx(aIdNum) );
   if ( ndx == scArr.end() )
      throw StudentCollectionXptn( "delete1Course",
                                   StudentCollectionXptn::notFound, aIdNum );
   (ndx->second).delete1Course( aCourse );
} // StudentCollection::delete1Course
```

Τέλος, για φύλαξη/φόρτωση έχουμε:

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

void StudentCollection::save( ofstream& bout, SIndexEntry* index )
{
   if ( bout.fail() )
      throw StudentCollectionXptn( "save",
                                   StudentCollectionXptn::fileNotOpen );
   size_t scNOfStudents( scArr.size() );
   bout.write( reinterpret_cast<const char*>(&scNOfStudents),
               sizeof(scNOfStudents) );
   for ( map<unsigned int,Student>::const_iterator it(scArr.begin()),
         unsigned int k(0);
         it != scArr.end();  ++it, ++k )
   {
      index[k].sIdNum = it->first;
      index[k].loc = bout.tellp();
      (it->second).save( bout );
   }
   if ( bout.fail() )
```

```
      throw StudentCollectionXptn( "save",
                                   StudentCollectionXptn::cannotWrite );
} // StudentCollection::save

void StudentCollection::load( ifstream& bin )
{
   StudentCollection tmp;
   size_t n;
   bin.read( reinterpret_cast<char*>(&n), sizeof(size_t) );
   if ( !bin.eof() )
   {
      for ( int k(0); k < n && !bin.fail(); ++k )
      {
         Student oneStudent;
         oneStudent.load( bin );
         tmp.add1Student( oneStudent );
      }
      if ( bin.fail() )
         throw StudentCollectionXptn( "load",
                                 StudentCollectionXptn::cannotRead );
      swapArr( tmp );
   }
} // StudentCollection::load
```

Φυσικά, θα αλλάξει και η *checkWH()*:

```
void StudentCollection::checkWH( ostream& log, int maxWH ) const
{
   if ( maxWH <= 0 )
      throw StudentCollectionXptn( "checkWH",
                                   StudentCollectionXptn::negWH, maxWH );
   for ( map<unsigned int,Student>::const_iterator it(scArr.begin());
         it != scArr.end();  ++it )
   {
      if ( (it->second).getWH() > maxWH )
         log << "student with id num " << (it->second).getIdNum()
             << ": " << (it->second).getWH() << " hours/week"
             << endl;
   } // for
} // StudentCollection::checkWH
```

Prj07.2.6 Κλάση *StudentInCourseCollection*

Εδώ η απεικόνισή μας θα είναι:

```
map< StudentInCourse::SICKey, StudentInCourse > siccArr;
```

Αυτό θα πει ότι θα βάλουμε μια φορά το κλειδί και άλλη μια φορά το κλειδί μαζί με μια τιμή `float`; Θα προτιμούσες δηλαδή να βάλουμε "`map<StudentInCourse::SICKey, float>`"; Αν θέλεις, δοκίμασέ το ως μια (εύκολη) άσκηση. Να έχεις υπόψη σου πάντως ότι – σε πραγματικές συνθήκες– μπορεί να υπάρχουν και άλλα στοιχεία εκτός από έναν βαθμό.

Η παραπάνω δήλωση προϋποθέτει την επιφόρτωση του τελεστή "`<`" για την κλάση *StudentInCourse*. Αντί να κάνουμε αυτό, κυρίως για εκπαιδευτικούς λόγους, θα ορίσουμε το συναρτησοειδές:

```
   struct SICKeyLT
   {
      bool operator()( const SICKey& frst, const SICKey& scnd ) const
      {
         bool fv( false );
         if ( frst.CCode < scnd.CCode )  fv = true;
         else if ( frst.CCode == scnd.CCode )
         {
            if ( frst.sIdNum < scnd.sIdNum )  fv = true;
         }
         return fv;
      }
```

```
   }; // SICKeyLT
```

θα ορίσουμε τη συντομογραφία *SICMap* ως:

```
   typedef map< StudentInCourse::SICKey, StudentInCourse,
                StudentInCourseCollection::SICKeyLT > SICMap;
```

και θα δηλώσουμε:

```
   SICMap siccArr;
```

Πώς θα αποθηκεύονται τα αντικείμενα στην απεικόνιση; Όλα τα αντικείμενα με τον ίδιο κωδικό μαθήματος θα αποθηκεύονται μαζί (κατ' αύξοντα αριθμό μητρώου).

Πού μπαίνουν όλα αυτά; Στην αρχή της περιοχής "**private**":

```
class StudentInCourseCollection
{
public:
// . . .
private:
   struct SICKeyLT
// ΟΠΩΣ ΠΑΡΑΠΑΝΩ
   typedef map< StudentInCourse::SICKey, StudentInCourse,
                StudentInCourseCollection::SICKeyLT > SICMap;
   SICMap siccArr;
// . . .
}; // StudentInCourseCollection
```

Σημείωση:▶

Ως άσκηση (εύκολη) δοκίμασε και τον άλλο δρόμο: επιφόρτωσε τον τελεστή "**<**" για την κλάση *StudentInCourse*:

```
bool operator<( const StudentInCourse::SICKey& frst,
                const StudentInCourse::SICKey& scnd )
{
   bool fv( false );
   if ( frst.CCode < scnd.CCode )  fv = true;
   else if ( frst.CCode == scnd.CCode )
   {
      if ( frst.sIdNum < scnd.sIdNum )  fv = true;
   }
   return fv;
} // operator<
```

Σύγκρινέ την με την επιφόρτωση του "**()**" στο συναρτησοειδές.◀

Μετά τους παραπάνω ορισμούς και δηλώσεις έχουμε δημιουργό:

```
StudentInCourseCollection::StudentInCourseCollection()
{
   siccPAllStudents = 0;
   siccPAllCourses = 0;
} // StudentInCourseCollection::StudentInCourseCollection
```

και καταστροφέα:

```
   ~StudentInCourseCollection()  { };
```

Ακόμη έχουμε:

```
   size_t getNOfStudentInCourses() const  {  return siccArr.size();  }

void StudentInCourseCollection::getArr( vector<StudentInCourse>& sicTbl) const
{
   try
   {
      for ( StudentInCourseCollection::SICMap::const_iterator
                                                    it(siccArr.begin());
            it != siccArr.end();  ++it )
         sicTbl.push_back( it->second );
   }
   catch( bad_alloc& )
   {  throw StudentCollectionXptn( "getArr",
                                   StudentCollectionXptn::allocFailed );  }
```

```
} // StudentInCourseCollection::getArr

StudentInCourseCollection::SICMap::iterator
                 StudentInCourseCollection::findNdx( int aIdNum, string code )
{
   StudentInCourseCollection::SICMap::iterator ndx;
   if ( aIdNum <= 0 || code.length() != Course::cCodeSz-1 )
      ndx = siccArr.end();
   else
      ndx = siccArr.find( StudentInCourse::SICKey(aIdNum, code) );
   return ndx;
} // StudentInCourseCollection::findNdx
```

Οι μέθοδοι για «1 *StudentInCourse*» γίνονται:

```
   bool find1StudentInCourse( int aIdNum, string code )
   {  return ( findNdx(aIdNum, code) != siccArr.end() );  }

const StudentInCourse&
          StudentInCourseCollection::get1StudentInCourse( int aIdNum,
                                                          string code ) const
{
   StudentInCourseCollection::SICMap::const_iterator
                                                 ndx( findNdx(aIdNum, code) );
   if ( ndx == siccArr.end() )
      throw StudentInCourseCollectionXptn( "get1StudentInCourse",
                                     StudentInCourseCollectionXptn::notFound,
                                     StudentInCourse::SICKey(aIdNum, code) );
   return ndx->second;
} // StudentInCourseCollection::get1StudentInCourse

void StudentInCourseCollection::delete1StudentInCourse( int aIdNum,
                                                        string code )
{
   StudentInCourseCollection::SICMap::iterator
                                      ndx( findNdx(aIdNum, code) );
   if ( ndx != siccArr.end() ) // 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 );
      siccArr.erase( ndx );
   }
} // StudentInCourseCollection::delete1StudentInCourse

void StudentInCourseCollection::add1StudentInCourse(
                                        const StudentInCourse& aStdnInCrs )
{
   if ( findNdx( aStdnInCrs.getSIdNum(),
                           aStdnInCrs.getCCode()) == siccArr.end() )
   {
      if ( siccPAllCourses == 0 )
         throw StudentInCourseCollectionXptn( "add1StudentInCourse",
                             StudentInCourseCollectionXptn::noCrs );
      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 );
      try
      {
        siccArr.insert( make_pair(aStdnInCrs.getKey(), aStdnInCrs));
      }
      catch( bad_alloc& )
      {  throw StudentInCourseCollectionXptn( "add1StudentInCourse",
                              StudentInCourseCollectionXptn::allocFailed );  }
   }
} // StudentInCourseCollection::add1StudentInCourse
```

Τέλος, οι *save()*, *load()* και *swapArr()* γίνονται:

```
void StudentInCourseCollection::save( ofstream& bout ) const
{
   if ( bout.fail() )
      throw  StudentInCourseCollectionXptn( "save",
                                    StudentInCourseCollectionXptn::fileNotOpen );
   size_t siccNOfStudentInCourses( siccArr.size() );
   bout.write( reinterpret_cast<const char*>(&siccNOfStudentInCourses),
               sizeof(siccNOfStudentInCourses) );
   for ( StudentInCourseCollection::SICMap::const_iterator
                                                        it(siccArr.begin());
         it != siccArr.end();  ++it )
      (it->second).save( bout );
   if ( bout.fail() )
      throw  StudentInCourseCollectionXptn( "save",
                                    StudentInCourseCollectionXptn::cannotWrite );
} // StudentInCourseCollection::save

void StudentInCourseCollection::load( ifstream& bin )
{
   StudentInCourseCollection tmp;
   size_t n;
   bin.read( reinterpret_cast<char*>(&n), sizeof(size_t) );
   if ( !bin.eof() )
   {
      for ( int k(0); k < n && !bin.fail(); ++k )
      {
         StudentInCourse oneStudentInCourse;
         oneStudentInCourse.load( bin );
         tmp.add1StudentInCourse( oneStudentInCourse );
      }
      if ( bin.fail() )
         throw  StudentInCourseCollectionXptn( "load",
                           StudentInCourseCollectionXptn::cannotRead );
      swapArr( tmp );
   }
} // StudentInCourseCollection::load

void StudentInCourseCollection::swapArr( StudentInCourseCollection& rhs )
{
   siccArr.swap( rhs.siccArr );
} // StudentInCourseCollection::swap
```

Prj07.2.6.1 Δυο Λόγια για τη *SICKeyLT*

Είπαμε παραπάνω ότι με το συναρτησοειδές που ορίσαμε «*Όλα τα αντικείμενα με τον ίδιο κωδικό μαθήματος θα αποθηκεύονται μαζί (κατ' αύξοντα αριθμό μητρώου).*» Ας το δούμε αυτό.

Βάζουμε στο πρώτο πρόγραμμα, πριν από τη φύλαξη των συλλογών, τις εντολές:

```
   log.open( "sic.txt" );
   vector<StudentInCourse> allSic;
   allEnrollments.getArr( allSic );
   for ( int k(0); k < allSic.size(); ++k )
      allSic[k].display( log );
   log.close();
```

Στο `sic.txt` βλέπουμε τα εξής:

```
2019 ΕΥ0100Ε 0
2099 ΕΥ0100Ε 0
2173 ΕΥ0100Ε 0
2069 ΕΥ0100Θ 0
2077 ΕΥ0100Θ 0
2081 ΕΥ0100Θ 0
2117 ΕΥ0100Θ 0
. . .
2067 ΤΠ0305Θ 0
2240 ΤΠ0305Θ 0
2044 ΤΠ0308Θ 0
2067 ΤΠ0314Ε 0
2059 ΤΠ0314Θ 0
2091 ΤΠ0314Θ 0
2127 ΤΠ0314Θ 0
2153 ΤΠ0314Θ 0
2164 ΤΠ0314Θ 0
2203 ΤΠ0314Θ 0
```

ή αλλιώς:

```
ΕΥ0100Ε
2019
2099
2173
ΕΥ0100Θ
2069
2077
2081
2117
. . .
ΤΠ0305Θ
2067
2240
2044
ΤΠ0314Ε
2067
ΤΠ0314Θ
2059
2091
2127
2153
2164
2203
```

Όπως βλέπεις, εδώ έχουμε τις καταστάσεις εγγεγραμμένων στα μαθήματα που δίνει η γραμματεία στους διδάσκοντες.

Καλό θα είναι να αλλάξεις τη σειρά των συγκρίσεων στο συναρτησοειδές κα να ξαναδοκιμάσεις. Μπορείς να προβλέψεις τι θα πάρεις; Τις δηλώσεις μαθημάτων ανά φοιτητή (όπως υπάρχουν στα αντικείμενα –τύπου *Student*– του μητρώου των φοιτητών).

Prj07.2.7 Πίνακας – Ευρετήριο

Στην §Prj07.2.1.5 είπαμε σχετικώς με τον πίνακα-ευρετήριο «*Δεν θα αλλάξουμε περιέχον αλλά θα ταξινομήσουμε τον πίνακα (κατ' αύξοντα αριθμό μητρώου) ώστε να υπάρχει δυνατότητα δυαδικής αναζήτησης από τα προγράμματα εφαρμογής που τον χρησιμοποιούν.*»

Ε, δεν θα χρειαστεί να κουραστούμε: ο πίνακας *είναι* ταξινομημένος όπως τον θέλουμε! Πράγματι, στην §Prj07.2.5, για την κλάση *StudentCollection*, δηλώσαμε περιέχον για τη συλλογή:

```
map< unsigned int, Student > scArr;
```

Το εννοούμενο τρίτο όρισμα του περιέχοντος είναι "`less<unsigned int>`". Στο scArr εισάγουμε στοιχεία με την

"`scArr.insert( make_pair(aStudent.getIdNum(),aStudent) )`"

και τέλος, για τη φύλαξη στο αρχείο, διασχίζουμε το περιέχον με τη:

```
for (map<unsigned int,Student>::const_iterator it(scArr.begin()),
     unsigned int k(0);
     it != scArr.end();  ++it, ++k )
{
   index[k].sIdNum = it->first;
   index[k].loc = bout.tellp();
   (it->second).save( bout );
}
```

Και πού μπαίνει η δυαδική αναζήτηση; Στο δεύτερο πρόγραμμα, αν θέλουμε. Κατ' αρχήν το δεύτερο πρόγραμμα –όπως και το πρώτο– δουλεύει μια χαρά χωρίς να χρειάζεται τροποποίηση.

Αν θέλεις να χρησιμοποιήσεις την (*std*::)*lower_bound()* για να εκμεταλλευθείς την ταξινόμηση του πίνακα θα πρέπει να κάνεις τα εξής:

- Ορίζεις την

```
bool comp( const SIndexEntry& frst, const SIndexEntry& scnd )
{  return ( frst.sIdNum < scnd.sIdNum );  } // comp
```

- Αντικαθιστάς τις:

```
int ndx( linSearch(sIndex, ndxSz, 0, ndxSz-1,
         SIndexEntry(idNum)) );
if ( ndx < 0 )
```

με τις

```
SIndexEntry* ndx( lower_bound(sIndex, sIndex+ndxSz,
                  SIndexEntry(idNum), comp) );
if ( ndx->sIdNum != idNum )
```

και τη

```
bin.seekg( sIndex[ndx].loc );
```

με τη

```
bin.seekg( ndx->loc );
```

Φυσικά, δεν θα ξεχάσεις να βάλεις και την "`#include <algorithm>`".

Prj07.3 Τι Είδαμε στα Δύο Παραδείγματα

Σύγκρινε τους δύο τρόπους που γράψαμε τις κλάσεις μας για να καταλάβεις ότι η άκριτη αποκλειστική χρήση του *vector* μπορεί να σου ευκολύνει κάπως τη δουλειά αλλά δεν σου δίνει το καλύτερο αποτέλεσμα.

Είναι ουσιώδες να κατανοήσεις καλά το πρόβλημα που έχεις να λύσεις και να επιλέξεις το καταλληλότερο περιέχον για την κάθε συλλογή δεδομένων.

Πάντως θα πρέπει να επισημάνουμε το πόσο καλό εργαλείο είναι το *vector* για την τεχνική RAII όταν ο πόρος που χειρίζεσαι είναι δυναμική μνήμη. Δηλαδή, τα άλλα περιέχοντα δεν είναι κατάλληλα; Μια χαρά είναι και τα άλλα αφού έχουν τους καταστροφείς τους αλλά το *vector* είναι το πιο απλό στον χειρισμό.

Τέλος, πρόσεξε ότι τα δικά μας εργαλεία (του `MyTmpltLib.h`) αχρηστεύθηκαν αφού η STL μας δίνει καλύτερες λύσεις.