project

3

Φοιτητές και Μαθήματα

Περιεχόμενα:

Prj03.1 Το Πρόβλημα

Μας δίνονται

- *ένα μη-μορφοποιημένο (binary) αρχείο με όνομα* **`gCourses.dta`** *με περιεχόμενο άγνωστο πλήθος αντικειμένων τύπου:*

```
class SylCourse
{
public:
   enum { cCodeSz = 8, cTitleSz = 80, cCategSz = 4 };
// . . .
private:
   char         cCode[cCodeSz];   // 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
   char         cPrereq[cCodeSz]; // προαπαιτούμενο
}; // SylCourse
```

που, όπως φαίνεται, περιγράφει ένα μάθημα προγράμματος σπουδών.

- *ένα μορφοποιημένο (text) αρχείο, με όνομα* **enrllmnt.txt**, *που περιέχει στοιχεία φοιτητών/τριών και δηλώσεων μαθημάτων στο τρέχον εξάμηνο. Ένα δείγμα περιεχομένου του αρχείου είναι:*

```
2149\tΜΠΡΟΥΜΟΥΤΗ\tΑΛΕΞΙΑ
4
ΕΥ0160Ε
ΕΥ02400
ΕΥ03610
ΤΠ02030

2059\tΜΥΛΩΝΑΣ\tΣΤΑΥΡΟΣ
6
ΕΥ02210
ΕΥ0240Ε
ΕΥ0331Ε
ΤΠ01010
ΤΠ0305Ε
ΤΠ03140
```

*Εδώ έχουμε στοιχεία και δηλώσεις μαθημάτων για δύο φοιτητές. Στην πρώτη γραμμή έχουμε τον αριθμό μητρώου του/της φοιτητή/τριας (π.χ. "*2149*" ή "*2059*"), μετά το επώνυμό του/της ("*ΜΠΡΟΥΜΟΥΤΗ*" ή "*ΜΥΛΩΝΑΣ*") και τέλος το όνομα ("*ΑΛΕΞΙΑ*" ή "*ΣΤΑΥΡΟΣ*"). Στην επόμενη γραμμή υπάρχει φυσικός αριθμός που είναι το πλήθος των μαθημάτων που δήλωσε ο/η φοιτητής/τρια. Στη συνέχεια ακολουθούν οι κωδικοί των μαθημάτων που δηλώθηκαν.*

Τα στοιχεία δύο φοιτητών διαχωρίζονται με μια κενή γραμμή.

Το αρχείο μαθημάτων είναι ελεγμένο. Το αρχείο δηλώσεων μπορεί να έχει λάθη δύο ειδών:

- *Περισσότερες από μια δηλώσεις του ίδιου φοιτητή.*
- *Λάθος κωδικό μαθήματος.*

Θέλουμε ένα πρόγραμμα που θα διαβάσει τα αρχεία που δίνονται και θα μας δώσει τέσσερα νέα:

- *Ένα μη-μορφοποιημένο αρχείο, με όνομα* **Students.dta**, *που θα έχει αντικείμενα τύπου*

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

ένα για κάθε φοιτητή.

- *Ένα μη-μορφοποιημένο αρχείο, με όνομα* **enrllmnt.dta**, *που θα έχει αντικείμενα τύπου*

```
class StudentInCourse
{
public:
// . . .
private:
   enum { sicCCodeSz = 8 };
   unsigned int sicSIdNum;            // αριθμός μητρώου
   char         sicCCode[sicCCodeSz]; // κωδικός μαθήματος
   float        sicMark;              // βαθμός στο μάθημα
}; // StudentInCourse
```

ένα για κάθε δήλωση (χωρίς λάθη) μαθήματος από φοιτητή.

- *Ένα μη-μορφοποιημένο αρχείο, με όνομα* **courses.dta**, *που θα έχει αντικείμενα τύπου Course, ένα για κάθε μάθημα. Η κλάση Course είναι σαν τη SylCourse με ένα επί πλέον μέλος:*

```
unsigned int cNoOfStudents;    // αριθ. φοιτητών στο μάθημα
```

Φυσικά όλα τα αντικείμενα θα έχουν ενημερωμένο το νέο μέλος.

- *Ένα μορφοποιημένο αρχείο καταγραφής λαθών, με όνομα* **log.txt**. *Εκτός των λαθών που περιγράψαμε παραπάνω θα καταγράφονται και οι αριθμοί μητρώου των φοιτητών που δήλωσαν μαθήματα με άθροισμα ωρών μεγαλύτερο των 30 ανά εβδομάδα.*

Σημείωση: ▶

Το αρχείο **gCourses.dta** *έχει γραφεί με την παρακάτω μέθοδο:*

```
void SylCourse::save( ostream& bout ) const
{
   if ( bout.fail() )
      throw SylCourseXptn( "save", SylCourseXptn::fileNotOpen );
   bout.write( cCode, sizeof(cCode) );                    // κωδικός μαθήματος
   bout.write( cTitle, sizeof(cTitle) );                   // τίτλος μαθήματος
   bout.write( reinterpret_cast<const char*>(&cFSem), sizeof(cFSem) );
                                                          // τυπικό εξάμηνο
   bout.write( reinterpret_cast<const char*>(&cCompuls), sizeof(cCompuls) );
                                                  // υποχρεωτικό ή επιλογής
   bout.write( &cSector, sizeof(cSector) );                       // τομέας
   bout.write( cCateg, sizeof(cCateg) );                      // κατηγορία
   bout.write( reinterpret_cast<const char*>(&cWH), sizeof(cWH) );
                                                  // ώρες ανά εβδομάδα
   bout.write( reinterpret_cast<const char*>(&cUnits), sizeof(cUnits) );
                                                 // διδακτικές μονάδες
   bout.write( cPrereq, sizeof(cPrereq) );                // προαπαιτούμενο
   if ( bout.fail() )
      throw SylCourseXptn( "save", SylCourseXptn::cannotWrite );
} // SylCourse::save
```

◀

Prj03.2 Το (Πρώτο) Σχέδιο για το Πρόγραμμα

Ας ξεκινήσουμε με δύο

Παρατηρήσεις ▶

1. Τα *cNoOfStudents* της *Course* και *sNoOfCourses*, *sWH* της *Student* φαίνονται κάπως «ξεκάφωτα». Και είναι! Όλα τα άλλα χαρακτηριστικά της *Course* είναι «πάγια» ενώ το *cNoOfStudents* αναφέρεται στο τρέχον εξάμηνο. Τα ίδια ισχύουν και για τα δύο χαρακτηριστικά της *Student*. Η σχεδίαση των κλάσεων δεν είναι σωστή! Παρ' όλα αυτά θα επιμείνουμε στη συγκεκριμένη περιγραφή για εκπαιδευτικούς λόγους: *Εδώ ενδιαφερόμαστε μόνο για την προγραμματιστική διαχείριση των κλάσεων.* Η σχεδίαση είναι αντικείμενο άλλων τομέων της τεχνολογίας λογισμικού.
2. Ένα αντικείμενο της *StudentInCourse* θα πρέπει να έχει όλες τις πληροφορίες που σχετίζονται με την εγγραφή ενός φοιτητή σε ένα μάθημα, π.χ.: δύο βαθμούς (για τις δύο εξεταστικές περιόδους) και όχι έναν, ακαδημαϊκό εξάμηνο (αν θέλουμε να έχουμε πολλά εξάμηνα) κλπ. Εδώ βάλαμε μόνον έναν βαθμό, για να απλουστεύσουμε τα πράγματα.◀

Η διαφορά αυτού του παραδείγματος από το «παράδειγμα της μπαταρίας» είναι ότι εκεί είχαμε μια κλάση και η διατύπωση του προβλήματος καθόριζε τις μεθόδους που έπρεπε να αναπτυχθούν. Εδώ, που δεν έχουμε τέτοια βοήθεια, πώς θα βρούμε τις μεθόδους που χρειαζόμαστε; Κάνουμε ένα σχέδιο του προγράμματος και από εκεί εντοπίζουμε τις ανάγκες μας. Υπάρχουν βέβαια ορισμένα σχεδόν απαραίτητα συστατικά για οποιαδήποτε κλάση που σχετίζονται με το πρόγραμμα στο οποίο θα χρησιμοποιηθεί. Θα τα μάθουμε στα επόμενα μαθήματα.

Ας κάνουμε λοιπόν ένα πρώτο σχέδιο για να δούμε τι γίνεται:

```
Φόρτωσε τον πίνακα των μαθημάτων
Μηδένισε το cNoOfStudents του κάθε μαθήματος
Άνοιξε το αρχείο enrllmnt.txt
do {
   Διάβασε τα στοιχεία ενός φοιτητή
   if ( δεν τελείωσε το αρχείο )
   {
      if ( υπάρχει στον πίνακα φοιτητών )
      {
         Αγνόησε τη δήλωση // υπάρχει ήδη άλλη δήλωση
         Γράψε στο log
      }
      else
      {
         Βάλε τον φοιτητή στον πίνακα φοιτητών
         Μηδένισε τα sNoOfCourses και sWH του φοιτητή
         for ( όλα τα μαθήματα που δήλωσε ο φοιτητής )
         {
            αναζήτησε τον κωδικό του μαθήματος
                                              στον πίνακα μαθημάτων
            if ( δεν υπάρχει )
               Γράψε στο log
            else
            {
               Βάλε το (ΑΜ φοιτητή, κωδικός μαθήματος)
                                               στον πίνακα δηλώσεων
               Ενημέρωσε τα sNoOfCourses και sWH του φοιτητή
               Ενημέρωσε το cNoOfStudents του κάθε μαθήματος
            }
         } // for
      } // if ( υπάρχει στον πίνακα φοιτητών )
   } // if (δεν τελείωσε το αρχείο )
} while ( δεν τελείωσε το αρχείο )
for ( όλους τους φοιτητές )
   if ( sWH > 30 ) Γράψε στο log
Φύλαξε τα στοιχεία του πίνακα φοιτητών στο Students.dta
Φύλαξε τα στοιχεία του πίνακα δηλώσεων στο enrllmnt.dta
Φύλαξε τα στοιχεία του πίνακα μαθημάτων στο gCourses.dta
```

Στη συνέχεια θα καταγράψουμε τις απαιτήσεις που προκύπτουν για κάθε κλάση.

Prj03.3 Η Κλάση *Course*

Τι πρέπει να κάνουμε με την *Course*;

- Να διαβάζουμε στοιχεία αντικειμένων της κλάσης από μη-μορφοποιημένο αρχείο («*Φόρτωσε τον πίνακα των μαθημάτων*»). Αν υλοποιήσουμε την κλάση *SylCourse* θα πρέπει να γράψουμε
  - μια μέθοδο, ας την πούμε *load*, που θα φορτώνει τα στοιχεία ενός μαθήματος από (μη-μορφοποιημένο) αρχείο και
  - μια μέθοδο (ή, καλύτερα, επιφόρτωση του τελεστή εκχώρησης) που θα αντιγράφει ένα αντικείμενο *SylCourse* σε ένα αντικείμενο *Course*.

  Κάτι τέτοιο όμως είναι έξω από τους στόχους μας.[1] Αντί για αυτό θα γράψουμε μια συνάρτηση –ας την πούμε *loadSylCourse*– που θα διαβάζει τις τιμές μελών του αντικειμένου *SylCourse* και θα τις περνάει μια προς μία στα αντίστοιχα μέλη ενός αντικειμένου *Course*. Αυτό το «πέρασμα τιμών» θα πρέπει να γίνει με μεθόδους που θα δούμε στη συνέχεια.

[1] Αργότερα θα υλοποιήσουμε την κλάση *SylCourse* (με όνομα *Course*).

- Να μπορούμε να μηδενίζουμε τον μετρητή («*Μηδένισε το cNoOfStudents του κάθε μαθήματος*»). Αυτό μπορεί να γίνει είτε με μια μέθοδο *clearStudents()* που κάνει μόνον αυτό είτε με μια *setNoOfStudents()* που είναι γενικότερη.
- Να βρίσκουμε ένα αντικείμενο με συγκεκριμένο κωδικό («*Αναζήτησε τον κωδικό του μαθήματος στον πίνακα μαθημάτων*»). Επειδή η αναζήτηση θα γίνεται με βάση τον κωδικό θα χρειαστούμε μια μέθοδο που να μας δίνει τον κωδικό (*getCode()*).
- Να παίρνουμε τις ώρες διδασκαλίας ανά εβδομάδα για να τις προσθέσουμε στις αντίστοιχες του φοιτητή («*Ενημέρωσε τα sNoOfCourses και sWH του φοιτητή*»). Θα πρέπει να γράψουμε μια *getWH*.
- Να αυξάνουμε τον αριθμό των φοιτητών που γράφονται στο μάθημα κατά 1 («*Ενημέρωσε το cNoOfStudents του κάθε μαθήματος*»). Μπορούμε είτε να γράψουμε μια *add1Student()* που αυξάνει την τιμή του *cNoOfStudents* κατά 1 είτε να γράψουμε μια *getNoOfStudents()* και να τη χρησιμοποιήσουμε μαζί με τη *setNoOfStudents()*.
- Να γράφουμε στοιχεία αντικειμένων της κλάσης σε μη-μορφοποιημένο αρχείο («*Φύλαξε τα στοιχεία του πίνακα μαθημάτων στο* `courses.dta`»). Ας πούμε *save* τη σχετική μέθοδο.

Ξεκινούμε με τη *loadSylCourse()*. Όπως μπορούμε να δούμε στη δήλωση της *SylCourse* έχουμε:

- Ένα μέλος τύπου `bool` (*cCompuls*).
- Τρία μέλη τύπου `unsigned int` (*cFSem*, *cWH*, *cUnits*).
- Πέντε μέλη τύπου `char` (*cCode* και *cPrereq* με μήκος *cCodeSz* (8), *cTitle* με μήκος *cTitleSz* (80), *cCateg* με μήκος *cCategSz* (4) και *cSector* με μήκος 1).

Αν λοιπόν δηλώσουμε τρεις ενταμιευτές:[2]

```
   bool bBuf;
   unsigned int iBuf;
   char cBuf[Course::cTitleSz];
```

μπορούμε να διαβάσουμε αυτά που γράφηκαν με τη *SylCourse*`::`*save* με τις εξής εντολές:

```
   bin.read( cBuf, Course::cCodeSz );                    // κωδικός μαθήματος
   bin.read( cBuf, Course::cTitleSz );                    // τίτλος μαθήματος
   bin.read( reinterpret_cast<char*>(&iBuf), sizeof(unsigned int) );
                                                          // τυπικό εξάμηνο
   bin.read( reinterpret_cast<char*>(&bBuf), sizeof(bool) );
                                                 // υποχρεωτικό ή επιλογής
   bin.read( cBuf, sizeof(char) );                          // τομέας
   bin.read( cBuf, Course::cCategSz );                   // κατηγορία
   bin.read( reinterpret_cast<char*>(&iBuf), sizeof(unsigned int) );
                                                 // ώρες ανά εβδομάδα
   bin.read( reinterpret_cast<char*>(&iBuf), sizeof(unsigned int) );
                                                 // διδακτικές μονάδες
   bin.read( cBuf, Course::cCodeSz );                    // προαπαιτούμενο
```

Η *loadSylCourse()* θα πρέπει να είναι:

```
void loadSylCourse( ifstream& bin, Course& oneCourse )
{
   bool bBuf;
   unsigned int iBuf;
   char cBuf[Course::cTitleSz];

   bin.read( cBuf, Course::cCodeSz );                    // κωδικός μαθήματος
   if ( !bin.eof() )
   {
                                                 oneCourse.setCode( cBuf );
```

[2] και με την προϋπόθεση ότι και στην *Course* θα δηλώσουμε:

```
public:
   enum { cCodeSz = 8, cTitleSz = 80, cCategSz = 4 };
```

```
      bin.read( cBuf, Course::cTitleSz );          oneCourse.setTitle( cBuf );
      bin.read( reinterpret_cast<char*>(&iBuf), sizeof(unsigned int) );
                                                   oneCourse.setFSem( iBuf );
      bin.read( reinterpret_cast<char*>(&bBuf), sizeof(bool) );
                                                  oneCourse.setCompuls( bBuf );
      bin.read( cBuf, sizeof(char) );           oneCourse.setSector( cBuf[0] );
      bin.read( cBuf, Course::cCategSz );          oneCourse.setCateg( cBuf );
      bin.read( reinterpret_cast<char*>(&iBuf), sizeof(unsigned int) );
                                                     oneCourse.setWH( iBuf );
      bin.read( reinterpret_cast<char*>(&iBuf), sizeof(unsigned int) );
                                                   oneCourse.setUnits( iBuf );
      bin.read( cBuf, Course::cCodeSz );          oneCourse.setPrereq( iBuf );
      if ( bin.fail() )
         throw ProgXptn( "loadSylCourse", ProgXptn::cannotRead );
   }
} // loadSylCourse
```

Όπως βλέπεις, θα χρειαστούμε μεθόδους *setCode*() *setFSem(), setCompuls(), setSector(), setCateg(), setWH(), setUnits()* και *setPrereq()* για να αλλάζουμε (καθορίζουμε) τις τιμές των *cCode, cTitle, cFSem, cCompuls, cSector, cCateg, cWH, cUnits* και *cPrereq* αντιστοίχως.

Αυτές οι μέθοδοι θα πρέπει να γραφούν προσεκτικά διότι δεν θα πρέπει να αφήνουν «σκουπίδια» να περνούν μέσα στα αντικείμενα. Αργότερα θα δούμε πώς κάνουμε αυτήν τη δουλειά με πιο συστηματικό τρόπο. Εδώ, έχοντας τη σιγουριά ότι διαβάζουμε ένα αρχείο με σωστά στοιχεία, θα γράψουμε μεθόδους που θα περνούν τα στοιχεία χωρίς ελέγχους. Αλλά, για να δεις πώς περίπου γίνονται αυτοί οι έλεγχοι, θα γράψουμε

- Τη *setFSem()* έτσι ώστε να δέχεται ως τυπικό εξάμηνο διδασκαλίας του μαθήματος κάτι μεταξύ 1 και 8.

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

- Τη *setPrereq()* έτσι ώστε να μην επιτρέπει ένα μάθημα να φέρεται ως προαπαιτούμενο του εαυτού του.

```
void Course::setPrereq( const string& prCode )
{
   if ( (prCode.length() > 0) && (prCode == cCode) )
      throw CourseXptn( "setPrereq", CourseXptn::autoRef, prCode.c_str() );
   strncpy( cPrereq, prCode.c_str(), cCodeSz-1 );  cPrereq[cCodeSz-1] = '\0';
} // Course::setPrereq
```

Φυσικά, για να διασφαλίσουμε ότι ένα μάθημα δεν φέρεται ως προαπαιτουμενο του εαυτού του θα πρέπει να προσέξουμε και τη

```
void Course::setCode( string aCode )
{
   if ( (aCode.length() > 0) && (aCode == cPrereq) )
      throw CourseXptn( "setCode", CourseXptn::autoRef, aCode.c_str() );
   strncpy( cCode, aCode.c_str(), cCodeSz-1 );        cCode[cCodeSz-1] = '\0';
} // Course::setCode
```

Οι υπόλοιπες, οι «απρόσεκτες» προς το παρόν, θα είναι:

```
void Course::setTitle( string aTitle )
{
   strncpy( cTitle, aTitle.c_str(), cTitleSz-1 );  cTitle[cTitleSz-1] = '\0';
} // Course::setTitle
void Course::setSector( char aSector )  {  cSector = aSector;  }
void Course::setCateg( string aCateg )
{
   strncpy( cCateg, aCateg.c_str(), cCategSz-1 );  cTitle[cCategSz-1] = '\0';
} // Course::setCateg
void Course::setWH( int aWH )  {  cWH = aWH;  }
```

```
void Course::setUnits( int aUnits )  {  cUnits = aUnits;  }
```

Ας δούμε τώρα τα υπόλοιπα· και αρχίζουμε από τα εύκολα:

```
   unsigned int getNoOfStudents() {  return cNoOfStudents;  }
   void clearStudents() {  cNoOfStudents = 0;  }
   void add1Student() {  ++cNoOfStudents;  }
   unsigned int getWH() const {  return cWH;  }
```

Γιατί δεν γράψαμε μια *setNoOfStudents*; Διότι αυτές που γράψαμε είναι πιο λειτουργικές: Αρχικώς, για κάποιο μάθημα "`Course ac`", βάζουμε:

```
   ac.clearStudents();
```

και κάθε φορά που βρίσκουμε φοιτητή που το έχει δηλώσει δίνουμε:

```
   ac.add1Student();
```

Αν υλοποιούσαμε τη *setNoOfStudents* θα είχαμε αντιστοίχως:

```
   ac.setNoOfStudents( 0 );
```

και:

```
   ac.setNoOfStudents( ac.getNoOfStudents()+1 );
```

Αλλά, η *setNoOfStudents* χρειάζεται έλεγχο (για αρνητικές τιμές) ενώ και οι μη αρνητικές είναι «προβληματικές» (τι νόημα έχει `ac.setNoOfStudents(20);`) Η *getNoOfStudents()* χρειάζεται γενικότερα.

Εύκολη είναι και η

```
   const char* getCode() const {  return cCode;  }
```

αλλά εδώ χρειάζονται εξηγήσεις για το πρώτο "`const`". Γιατί χρειάζεται; Βγάλε τα δύο "`const`" και δοκίμασε τα παρακάτω:

```
   cout << ac.getCode() << endl;
   char* p( ac.getCode() );
   p[3] = 'X';  p[4] = 'Z';
   cout << ac.getCode() << endl;
```

Αποτέλεσμα:

```
EY0140E
EY0XZ0E
```

Δηλαδή, ανοίξαμε μια «κερκόπορτα» από όπου τροποποιείται αυθαιρέτως το αντικείμενό μας.

Κρατώντας τα δύο "`const`" (που πάνε «πακέτο») το μόνο που μπορείς να κάνεις με το *p* είναι:

```
   char* p( new char[strlen(ac.getCode())+1] );
   strcpy( p, ac.getCode() );
```

Τώρα ο *p* δείχνει *αντίγραφο* του *ac.cCode* και δεν έχει σχέση με το αντικείμενό μας.

Ας έλθουμε τώρα στις *load()* και *save()*. Η *save()* θα είναι σαν τη *SylCourse::save()*, που μας δόθηκε, αλλά θα φυλάγει και την τιμή του *cNoOfStudents*:

```
void Course::save( ostream& bout ) const
{
   if ( bout.fail() )
      throw CourseXptn( "save", CourseXptn::fileNotOpen );
   bout.write( cCode, sizeof(cCode) );                    // κωδικός μαθήματος
   bout.write( cTitle, sizeof(cTitle) );                   // τίτλος μαθήματος
   bout.write( reinterpret_cast<const char*>(&cFSem), sizeof(cFSem) );
                                                          // τυπικό εξάμηνο
   bout.write( reinterpret_cast<const char*>(&cCompuls), sizeof(cCompuls) );
                                                 // υποχρεωτικό ή επιλογής
   bout.write( &cSector, sizeof(cSector) );                        // τομέας
   bout.write( cCateg, sizeof(cCateg) );                        // κατηγορία
   bout.write( reinterpret_cast<const char*>(&cWH), sizeof(cWH) );
                                                       // ώρες ανά εβδομάδα
   bout.write( reinterpret_cast<const char*>(&cUnits), sizeof(cUnits) );
                                                    // διδακτικές μονάδες
   bout.write( cPrereq, sizeof(cPrereq) );                 // προαπαιτούμενο
```

```
   bout.write( reinterpret_cast<const char*>(&cNoOfStudents),
               sizeof(cNoOfStudents) );              // αριθ. φοιτ. στο μάθημα
   if ( bout.fail() )
      throw CourseXptn( "save", CourseXptn::cannotWrite );
} // Course::save
```

Η *load()* θα γραφεί με οδηγό τη *save*, αφού θα διαβάσουμε όπως γράψαμε:

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

Αυτή είναι η *load()* αλλά δεν είναι καλή. Γιατί; Όταν δίνεις την εντολή:

```
   ac.load( bin );
```

το αντικείμενο *ac* έχει κάποια τιμή. Αν κάτι «πάει στραβά» κατά την ανάγνωση αυτή η τιμή καταστρέφεται. Θα πρέπει να διαβάζουμε πιο προσεκτικά, όπως θα μάθουμε αργότερα. Προς το παρόν δεν θα έχουμε πρόβλημα, όπως θα δεις στη συνέχεια.

Για να μπορέσεις να κάνεις δοκιμές με την κλάση την εξοπλίζουμε και με μια:

```
void Course::display( ostream& tout ) const
{
   if ( tout.fail() )
      throw CourseXptn( "display", CourseXptn::fileNotOpen );
   tout << cCode << '\t' << cTitle << '\t';
   if ( cCompuls )  tout << "Y\t";
              else  tout << "YE\t";
   tout << cSector << '\t' << cCateg << '\t' << cWH << '\t'
        << cUnits << '\t' << cPrereq << '\t' << cFSem << '\t'
        << cNoOfStudents << endl;
   if ( tout.fail() )
      throw CourseXptn( "display", CourseXptn::cannotWrite );
} // Course::display
```

Να λοιπόν η «προχειρογραμμένη»

```
class Course
{
public:
   enum { cCodeSz = 8, cTitleSz = 80, cCategSz = 4 };
   Course( string aCode="" );
   const char* getCode() const {  return cCode;  }
   unsigned int getWH() const {  return cWH;  }
   unsigned int getNoOfStudents() const {  return cNoOfStudents; }
   void setCode( string aCode );
   void setTitle( string aTitle );
   void setFSem( int aFSem );
   void setCompuls( bool aCompuls ) { cCompuls = aCompuls; };
   void setSector( char aSector )  {  cSector = aSector;  };
   void setCateg( string aCateg );
```

```
   void setWH( int aWH )  {  cWH = aWH;  };
   void setUnits( int aUnits )  {  cUnits = aUnits;  };
   void setPrereq( const string& prCode );
   void clearStudents() {  cNoOfStudents = 0;  }
   void add1Student() {  ++cNoOfStudents;  }
   void save( ostream& bout ) const;
   void load( istream& bin );
   void display( ostream& tout ) const;
private:
   char         cCode[cCodeSz];   // 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
   char         cPrereq[cCodeSz]; // προαπαιτούμενο
   unsigned int cNoOfStudents;    // αριθ. φοιτητών
}; // Course
```

και η αντίστοιχη κλάση εξαιρέσεων:

```
struct CourseXptn
{
   enum {  rangeError, autoRef,
           fileNotOpen, cannotWrite, cannotRead  };
   char funcName[100];
   int  errorCode;
   int  errIntVal;
   char errStrVal[100];
   CourseXptn( const char* mn, int ec, const char* sv="" )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';
      errorCode = ec;
      strncpy( errStrVal, sv, 99 );  errStrVal[99] = '\0';  }
   CourseXptn( const char* mn, int ec, int iv )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';
      errorCode = ec;
      errIntVal = iv;  }
}; // CourseXptn
```

Χρήσιμο είναι να ορίσουμε:

```
typedef Course* PCourse;
```

Prj03.4 Η Κλάση *Student*

Να δούμε τώρα τι απαιτήσεις έχουμε για τη *Student*.

- Η «*Διάβασε τα στοιχεία ενός φοιτητή*» παραπέμπει σε διάφορες εναλλακτικές λύσεις:
  - Να γράψουμε μια *readFromText* που διαβάζει τα στοιχεία από αρχείο –σαν τη *load*– αλλά, αυτήν τη φορά, μορφοποιημένο (text) ή
  - να γράψουμε τρεις "*set*": *setIdNum*, *setSurname*, *setFirstname* ή
  - να γράψουμε έναν δημιουργό

```
Student( int aIdNum=0, string aSurn="", string aFirstn="" )
```

- Για να κάνουμε τον έλεγχο «`if (`*υπάρχει στον πίνακα φοιτητών*`)`» πρέπει να αναζητούμε ένα αντικείμενο κλάσης *Student*. Επειδή η αναζήτηση θα γίνεται με βάση τον αριθμό μητρώου, θα χρειαστούμε μια μέθοδο που θα μας τον δίνει (*getIdNum*). Ανάγκη για την ίδια μέθοδο προκύπτει και από την «*Βάλε το (ΑΜ φοιτητή, κωδικός μαθήματος) στον πίνακα δηλώσεων*». Θα χρειαστούμε και εδώ έναν δημιουργό για αναζήτηση με τη *linSearch()*; Όχι, διότι στην περίπτωση αυτήν αναζητούμε ένα αντικείμενο που έχουμε διαβάσει από το αρχείο. Πάντως η επιφόρτωση του "`!=`" είναι αναγκαία.

- Ακολουθώντας το σκεπτικό που αναπτύξαμε για τη διαχείριση του μέλους *cNoOfStudents* της *Course*, από την «*Μηδένισε τα sNoOfCourses και sWH του φοιτητή*» συνάγουμε ανάγκη για μέθοδο *clearCourses()*.
- Για να ενημερώσουμε τους δύο μετρητές («*Ενημέρωσε τα sNoOfCourses και sWH του φοιτητή*») θα χρειαστούμε μέθοδο *add1Course()*.
- Για να ελέγξουμε τις εβδομαδιαίες ώρες του φοιτητή («`if (sWH > 30)`») θα χρειαστούμε μια *getWH()*.
- Τέλος, θα χρειαστούμε μια μέθοδο, ας την πούμε *save()*, για να γράφουμε στοιχεία αντικειμένων της κλάσης σε μη-μορφοποιημένο αρχείο («*Φύλαξε τα στοιχεία του πίνακα φοιτητών στο* `Students.dta`»).

  Ξεκινώντας και εδώ από τα εύκολα, έχουμε:

```
   unsigned int getIdNum() const {  return sIdNum;  }
   unsigned int getWH() const {  return sWH;  }
   void clearCourses() {  sNoOfCourses = 0;  sWH = 0;  }
   void add1Course( int aIncrWH );
```

όπου:

```
void Student::add1Course( int aIncrWH )
{
   if ( aIncrWH <= 0 )
      throw StudentXptn( "add1Course", StudentXptn::nonPosIncr,
                         aIncrWH );
   ++sNoOfCourses;
   sWH += aIncrWH;
} // Student::add1Course
```

Ο αριθμός μητρώου είναι ένα άλλο παράδειγμα *υποκατάστατου κλειδιού*. Μπορούμε λοιπόν να επιφορτώσουμε τον "`!=`" με σύγκριση αριθμών μητρώου μόνο:

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

Για το διάβασμα θα επιλέξουμε –για εκπαιδευτικούς λόγους, για να δεις τη διαφορά από τη *load()*– να γράψουμε μια:

```
void Student::readFromText( istream& tin )
{
   string line;
   getline( tin, line, '\n' );
   if ( !tin.eof() )
   {
      size_t t1Pos( line.find("\t") );
      if ( t1Pos >= line.length() )
         throw StudentXptn( "readFromText",
                            StudentXptn::incomplete );
      size_t t2Pos( line.find("\t", t1Pos+1) );
      if ( t2Pos >= line.length() )
         throw StudentXptn( "readFromText",
                            StudentXptn::incomplete );
      string str1( line.substr(0, t1Pos) );
      int    iStr1;
      iStr1 = atoi( str1.c_str() );
      if ( iStr1 <= 0 )
         throw StudentXptn( "readFromText",
                            StudentXptn::negIdNum, iStr1 );
      sIdNum = iStr1;
      str1 = line.substr( t1Pos+1, t2Pos-t1Pos-1 );
      strncpy( sSurname, str1.c_str(), sNameSz-1 );
                                       sSurname[sNameSz-1] = '\0';
      str1 = line.substr( t2Pos+1 );
      strncpy( sFirstname, str1.c_str(), sNameSz-1 );
                                    sFirstname[sNameSz-1] = '\0';
      sNoOfCourses = 0;
      sWH = 0;
```

```
   } // if ( !tin.eof . . .
} // Student::readFromText
```

Εδώ πρόσεξε τα εξής:

- Αν διαβάσουμε γραμμή που δεν έχει δύο '\t' ρίχνουμε εξαίρεση με κωδικό σφάλματος "*incomplete*". Γιατί το βάλαμε αυτό; Αφού έχουμε τη διαβεβαίωση ότι δεν υπάρχουν λάθη τέτοιου είδους. Η εμβέλεια της μεθόδου είναι πέρα από την τρέχουσα εφαρμογή.
- Ένα πρόβλημα μπορεί να παρουσιαστεί είναι το εξής: να πάρουμε όνομα ή επώνυμο που δεν χωράει στον πίνακα που έχουμε προβλέψει. Αλλά, σε τέτοια περίπτωση, δεν υπάρχει λόγος να ρίξουμε εξαίρεση. Απλώς αποθηκεύουμε όσο χωράει.

Η *save()* γράφεται όπως αυτή της *Course*:

```
void Student::save( ostream& bout ) const
{
   if ( bout.fail() )
      throw StudentXptn( "save", StudentXptn::fileNotOpen );
   bout.write( reinterpret_cast<const char*>(&sIdNum),
               sizeof(sIdNum) );                          // αριθμός μητρώου
   bout.write( sSurname, sNameSz );
   bout.write( sFirstname, sNameSz );
   bout.write( reinterpret_cast<const char*>(&sNoOfCourses),
               sizeof(sNoOfCourses) );              // αριθ. μαθημ. που δήλωσε
   bout.write( reinterpret_cast<const char*>(&sWH), sizeof(sWH) );
                                                        // ώρες ανά εβδομάδα
   if ( bout.fail() )
      throw StudentXptn( "save", StudentXptn::cannotWrite );
} // Student::save
```

Η *display()* μπορεί να είναι χρήσιμη και εδώ:

```
void Student::display( ostream& tout )
{
   tout << sIdNum << '\t' << sSurname << '\t' << sFirstname
        << endl << sNoOfCourses << '\t' << sWH << endl;
} // Student::display
```

Καταλήγουμε λοιπόν στην:

```
class Student
{
public:
   unsigned int getIdNum() const {  return sIdNum;  }
   void clearCourses() {  sNoOfCourses = 0;  sWH = 0;  }
   unsigned int getWH() const {  return sWH;  }
   void add1Course( int aIncrWH );
   void readFromText( istream& tin );
   void save( ostream& bout ) const;
   void display( ostream& tout );
private:
   enum { sNameSz = 20 };
   unsigned int sIdNum;        // αριθμός μητρώου
   char         sSurname[sNameSz];
   char         sFirstname[sNameSz];
   unsigned int sWH;           // ώρες ανά εβδομάδα
   unsigned int sNoOfCourses;  // αριθμός μαθημάτων που δήλωσε
}; // Student
```

και στην αντίστοιχη κλάση εξαιρέσεων:

```
struct StudentXptn
{
   enum { incomplete, negIdNum, nonPosIncr,
          fileNotOpen, cannotRead, cannotWrite };
   char funcName[100];
   int  errorCode;
   int  errValue;
   StudentXptn( char* mn, int ec, int ev = 0 )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';
      errorCode = ec;  errValue = ev;  }
```

```
}; // StudentXptn
```

Και εδώ θα ορίσουμε:

```
typedef Student* PStudent;
```

Prj03.5 Η Κλάση *StudentInCourse*

Για την κλάση αυτήν έχουμε:

- «*Βάλε το (ΑΜ φοιτητή, κωδικός μαθήματος) στον πίνακα δηλώσεων.*» Αυτή η απαίτηση μπορεί να αντιμετωπισθεί με έναν δημιουργό, με δύο "*set*" ή με μια "*set*".
- «*Φύλαξε τα στοιχεία του πίνακα δηλώσεων στο* `enrllmnt.dta`.» Από αυτό προκύπτει ανάγκη για μια *save*.

Εδώ, η *save()* είναι εύκολη:

```
void StudentInCourse::save( ostream& bout ) const
{
   if ( bout.fail() )
      throw StudentInCourseXptn( "save", StudentInCourseXptn::fileNotOpen );
   bout.write( reinterpret_cast<const char*>(&sicSIdNum), sizeof(sicSIdNum) );
                                                          // αριθμός μητρώου
   bout.write( sicCCode, sicCCodeSz );                   // κωδικός μαθήματος
   bout.write( reinterpret_cast<const char*>(&sicMark), sizeof(sicMark) );
                                                          // βαθμός στο μάθημα
   if ( bout.fail() )
      throw StudentInCourseXptn( "save",
                                 StudentInCourseXptn::cannotWrite );
} // StudentInCourse::save
```

Ας δούμε τώρα την πρώτη απαίτηση. Για να κάνουμε εισαγωγή ενός ζεύγους στον πίνακα θα πρέπει να έχουμε τη σιγουριά ότι *υπάρχουν και ο φοιτητής και το μάθημα*. Αυτό

- εξασφαλίζεται με αναζήτηση στους αντίστοιχους πίνακες και
- γίνεται με *μια* μέθοδο.

Επιπλέον, αυτή η μέθοδος θα πρέπει να ενημερώνει και τα αντίστοιχα αντικείμενα φοιτητή και μαθήματος.

Άρα οι επιλογές μας περιορίζονται σε δύο: ένας δημιουργός ή μια "*set*". Και οι δύο είναι σωστές, αλλά δεν είναι και τόσο «κομψό» σε δημιουργό μιας κλάσης να περνάς πίνακες με στοιχεία δύο άλλων κλάσεων και να ενημερώνει αντικείμενα άλλων κλάσεων. Για τον λόγο αυτόν επιλέγουμε τη «διπλή» "*set*":

```
void StudentInCourse::setIdNumCCode( unsigned int aIdNum,
                                     string aCCode,
                                     Student stArr[], int nOfStudents,
                                     Course crsArr[], int nOfCourses )
```

Τώρα βέβαια, παρατηρώντας το σχέδιο που κάναμε βλέπουμε ότι όταν έρχεται η ώρα να κάνουμε κάτι τέτοια ξέρουμε ότι ο φοιτητής υπάρχει σίγουρα. Επομένως μπορούμε να περάσουμε στη μέθοδο το αντικείμενο με τα στοιχεία του φοιτητή. Να λοιπόν μια απλοποιημένη μορφή της μεθόδου:

```
void StudentInCourse::setIdNumCCode( Student& aStudent,
                                     string aCCode,
                                     Course crsArr[], int nOfCourses )
{
   crsArr[nOfCourses].setCode( aCCode.c_str() );
   int cPos( 0 );
   while ( strcmp(crsArr[nOfCourses].getCode(),
                  crsArr[cPos].getCode())!=0 )
      ++cPos;
   if ( cPos == nOfCourses )
     throw StudentInCourseXptn( "StudentInCourse",
                                StudentInCourseXptn::unknownCCode,
                                aCCode.c_str() );
```

```
   sicSIdNum = aStudent.getIdNum();
   aStudent.add1Course( crsArr[cPos].getWH() );
   strcpy( sicCCode, aCCode.c_str() );
   crsArr[cPos].add1Student();
} // StudentInCourse::setIdNumCCode
```

Η αναζήτηση μαθήματος με βάση τον κωδικό γίνεται με γραμμική αναζήτηση με φρουρό, αλλά η τοποθέτηση του φρουρού δημιουργεί μια ανάγκη για την κλάση *Course*: χρειαζόμαστε μέθοδο *Course*::*setCode*.

Έτσι, η κλάση γίνεται:

```
class StudentInCourse
{
public:
   void setIdNumCCode( Student& aStudent, string aCCode,
                       Course crsArr[], int nOfCourses );
   void save( ostream& bout ) const;
   void display( ostream& tout ) const;
private:
   unsigned int sicSIdNum;      // αριθμός μητρώου
   char         sicCCode[8];    // κωδικός μαθήματος
}; // StudentInCourse
```

όπου:

```
void StudentInCourse::display( ostream& tout ) const
{
   tout << sicSIdNum << ' ' << sicCCode << endl;
} // StudentInCourse::display
```

Η κλάση εξαιρέσεων είναι:

```
struct StudentInCourseXptn
{
   enum { unknownCCode };
   char funcName[100];
   int  errorCode;
   char errStrVal[100];
   StudentInCourseXptn( const char* mn, int ec,
                        const char* sv="" )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';
      errorCode = ec;
      strncpy( errStrVal, sv, 99 );  errStrVal[99] = '\0';  }
}; // StudentInCourseXptn
```

Και εδώ θα ορίσουμε:

```
typedef StudentInCourse* PStudentInCourse;
```

Prj03.6 Το Πρόγραμμα

Ας δούμε τώρα πώς θα υλοποιήσουμε το σχέδιο προγράμματος που κάναμε. Και πρώτα η «*Φόρτωσε τον πίνακα των μαθημάτων.*» Αλλά, πώς θα γίνει η «φόρτωση»; Η φύλαξη έγινε με τη *save* που είδαμε παραπάνω. Αυτό σημαίνει ότι και η φόρτωση θα γίνει με τη *load* (αργότερα θα εξετάσουμε και άλλες δυνατότητες.)

Δηλώνουμε λοιπόν στο πρόγραμμά μας (στη **main**):

```
   Course*      crsArr;
   unsigned int nOfCourses;
```

και στη συνέχεια καλούμε:

```
   loadCourses( "gCourses.dta", crsArr, nOfCourses );
```

Prj03.6.1 Η *loadCourses()*

Τι θα κάνει η *loadCourses()*; Θα πάρει την απαραίτητη μνήμη, θα διαβάσει το αρχείο και θα αποθηκεύσει αυτά που διάβασε στον πίνακα crsArr. Αν κάτι δεν πάει καλά (αν δεν ανοίγει το αρχείο, αν δεν πάρουμε δυναμική μνήμη), θα έχουμε crsArr == 0 και *nOfCourses* == 0 και θα ρίξει εξαίρεση. Δηλαδή:

```
crsArr = 0;   nOfCourses = 0;
Πάρε μνήμη
if ( αποτυχία )  throw ...
Άνοιξε το αρχείο
if ( αποτυχία )
{  επίστρεψε τη μνήμη;  throw ...  }
while ( δεν τελείωσε το αρχείο )
{
   if ( δεν έχεις μνήμη )
      Πάρε και άλλη μνήμη (renew)
   Διάβασε ένα μάθημα
   Μηδένισε το cNoOfStudents του κάθε μαθήματος
   ++nOfCourses;
} // while
```

Στο αρχείο `MyTmpltLib.h` υπάρχει το (γνωστό μας) περίγραμμα

```
// renew - παίρνει μνήμη για πίνακα με nf στοιχεία τύπου T και
//         στα πρώτα ni (<= nf) στοιχεία του αντιγράφουμε αρχικώς
//         αυτά των ni θέσεων του αρχικού p.
template< typename T >
void renew( T*& p, int ni, int nf )
```

και θα το χρησιμοποιήσουμε. Η *renew()* όμως κάνει αντιγραφή του πίνακα κάθε φορά που τον «μεγαλώνει». Για να αποφύγουμε τις πολλές αντιγραφές δεν θα παίρνουμε ένα επιπλέον στοιχείο κάθε φορά αλλά *cIncr* (=10) στοιχεία.[3] Σε μια μεταβλητή, *resCourses*, κρατούμε τον αριθμό στοιχείων που έχουμε δεσμεύσει. Έτσι, η «*Πάρε μνήμη*» γίνεται:

```
   const int cIncr( 10 );
   unsigned int resCourses;    // θέσεις που έχουμε δεσμεύσει
   crsArr = 0;   nOfCourses = 0;
   try {  crsArr = new Course[cIncr];  }
   catch( bad_alloc )
   {  throw ProgXptn( "loadCourses", ProgXptn::allocFailed );  }
   resCourses = cIncr;
```

και η «*if (δεν έχεις μνήμη) πάρε και άλλη μνήμη*»:

```
      if ( nOfCourses+1 == resCourses )
      {
         renew( crsArr, nOfCourses, resCourses+cIncr );
         resCourses += cIncr;
      }
```

Εκείνο το "+1", στην *nOfCourses*+1 == *resCourses*, το χρειαζόμαστε για να μπορούμε να κάνουμε γραμμική αναζήτηση με φρουρό.

Να ολόκληρη η *loadCourses()*:

```
void loadCourses( string flNm,
                  Course*& crsArr, unsigned int& nOfCourses )
{
// Πάρε (λίγη) μνήμη
   const int cIncr( 10 );
   unsigned int resCourses;
   crsArr = 0;   nOfCourses = 0;
   try {  crsArr = new Course[cIncr];  }
   catch( bad_alloc )
   {  throw ProgXptn( "loadCourses", ProgXptn::allocFailed );  }
   resCourses = cIncr;
// Άνοιξε το αρχείο
```

[3] Αν θέλεις να κάνεις κάτι καλύτερο ξαναδιάβασε αυτά που είπαμε στην §16.13.3.

```
   istream bin( flNm.c_str(), ios_base::binary );
   if ( bin.fail() )
   {  delete[] crsArr;   // επίστρεψε τη μνήμη
      throw ProgXptn( "loadCourses", ProgXptn::cannotOpen, flNm.c_str() );
// Τώρα διάβαζε
   loadSylCourse( bin, crsArr[nOfCourses] );
   while ( !bin.eof() )
   {
      ++nOfCourses;
      crsArr[nOfCourses-1].clearStudents();
      if ( nOfCourses+1 == resCourses )
      {
         renew( crsArr, nOfCourses, resCourses+cIncr );
         resCourses += cIncr;
      } // if
      loadSylCourse( bin, crsArr[nOfCourses] );
   } // while
   bin.close();
} // loadCourses
```

Παρατηρήσεις ▶

- Πώς γίνεται το διάβασμα: διαβάζουμε αποθηκεύοντας στο πρώτο στοιχείο μετά το τελευταίο του πίνακα: στο `crsArr[nOfCourses]` (το οποίο υπάρχει πάντοτε). Αν όλα πάνε καλά (αν δεν ρίξει εξαίρεση η *load()*) αυξάνουμε τον *nOfCourses* (`++nOfCourses`) και το νέο στοιχείο περιλαμβάνεται στον πίνακα.
- Αν η *load()* δεν ρίξει εξαίρεση, για κάθε μάθημα που διαβάζουμε, μηδενίζουμε και τον *cNoOfStudents*.
- Η `catch` θα πιάσει οποιαδήποτε *CourseXptn*. Αν ο τύπος του λάθους δεν είναι αυτός που μας ενδιαφέρει (*cannotRead* ή *fileNotOpen*) την ξαναρίχνουμε (`throw`).◀

Prj03.6.2 Η Ανάγνωση του Αρχείου των Δηλώσεων

Η ανάγνωση του αρχείου `enrllmnt.txt` θα «γεμίσει» τον πίνακα φοιτητών και τον πίνακα δηλώσεων φοιτητών σε μαθήματα. Σε μεγάλο ποσοστό θα δουλέψουμε όπως δουλέψαμε παραπάνω. Κατ' αρχήν οι δηλώσεις:

```
   Student*    stArr;   unsigned int nOfStudents( 0 );
   const int sIncr( 10 );
   unsigned int resStudents;

   StudentInCourse* sic; unsigned int nOfStdInCrs( 0 );
   const int sicIncr( 30 );
   unsigned int resStdInCrs;
```

και οι πρώτες «φέτες» δυναμικής μνήμης:

```
      try { stArr = new Student[sIncr]; }
      catch( bad_alloc )
      {  throw ProgXptn( "main", ProgXptn::allocFailed );  }
      resStudents = sIncr;
      try { sic = new StudentInCourse[scIncr]; }
      catch( bad_alloc )
      {  throw ProgXptn( "main", ProgXptn::allocFailed );  }
      resStdInCrs = sicIncr;
```

Η «*Άνοιξε το αρχείο enrllmnt.txt*» γίνεται:

```
      ifstream tin( "enrllmnt.txt" );
      if ( tin.fail() )
         throw ProgXptn( "main", ProgXptn::cannotOpen, "enrllmnt.txt" );
```

Πριν από αυτό όμως θα πρέπει να ανοίξουμε το αρχείο καταγραφής προβλημάτων (*log*):

```
      log.open( "log.txt" );
      if ( log.fail() )
```

```
         throw ProgXptn( "main",
                         ProgXptn::cannotOpen, "log.txt" );
```

Prj03.6.3 Τα Στοιχεία Ενός Φοιτητή . . .

Η «*Διάβασε τα στοιχεία ενός φοιτητή*» είναι η *readFromText()*; Όχι! Πέρα από τη *readFromText()* θα πρέπει το αντικείμενο που θα διαβαστεί να εισαχθεί στον stArr με ό,τι συνεπάγεται (δυναμική μνήμη.)

```
void readAStudent( istream& tin,
                   PStudent& stArr, unsigned int& nOfStudents,
                   unsigned int& resStudents, int sIncr )
{
   if ( nOfStudents+1 == resStudents )
   {
      renew( stArr, nOfStudents, resStudents+sIncr );
      resStudents += sIncr;
   }
   stArr[nOfStudents].readFromText( tin );
   ++nOfStudents;
   stArr[nOfStudents-1].clearNoOfCourses();
   stArr[nOfStudents-1].clearWH();
} // readAStudent
```

Όπως βλέπεις διαβάζοντας έναν φοιτητή κάνουμε περίπου τα ίδια με αυτά που κάνουμε διαβάζοντας ένα μάθημα. Εδώ όμως υπάρχει μια διαφορά: Υπάρχει πιθανότητα να έχουμε διαβάσει ήδη δήλωση μαθημάτων του φοιτητή. Αυτό

- Γίνεται αντιληπτό από την ύπαρξη αντικειμένου-φοιτητή πιο πριν στον πίνακα μετά από αναζήτηση με τον αριθμό μητρώου.
- Αντιμετωπίζεται με το να αγνοήσουμε την τελευταία δήλωση.

```
         readAStudent( tin,
                       stArr, nOfStudents, resStudents, sIncr );
         if ( !tin.eof() )
         {
            int pos( 0 );
            while ( stArr[pos].getIdNum() !=
                    stArr[nOfStudents-1].getIdNum() ) ++pos;
            if ( pos < nOfStudents-1 )  // υπάρχει ήδη
            {// αγνόησε αυτά που ακολουθούν για τον φοιτητή
```

Τι ακολουθεί για τον φοιτητή και πώς το αγνοούμε;

- Ακολουθεί μια γραμμή με έναν αριθμό, ας τον πούμε *noc*, και
- *noc* γραμμές με έναν κωδικό μαθήματος στην κάθε μια.

Η παρακάτω συνάρτηση τα αγνοεί (τα διαβάζει και τα «ξεχνάει»):

```
void ignoreStudentData( istream& tin )
{
   string str1;
   getline( tin, str1, '\n' );
   int noc( atoi(str1.c_str()) );
   for ( int k(0); k < noc; ++k )  getline( tin, str1, '\n' );
} // ignoreStudentData
```

Συμπληρώνουμε λοιπόν:

```
            if ( pos < nOfStudents-1 )  // υπάρχει ήδη
            {// αγνόησε αυτά που ακολουθούν για τον φοιτητή
               --nOfStudents;
               ignoreStudentData( tin );
               log << " multiple entry for student with id num "
                   << stArr[pos].getIdNum() << endl;
            }
            else  // δεν υπάρχει
            { . . .
```

Prj03.6.4 . . . Και οι Δηλώσεις Μαθημάτων

Και αν δεν υπάρχει; Ε, τότε ακολουθούμε το σχέδιο:

```
for ( όλα τα μαθήματα που δήλωσε ο φοιτητής )
{
   αναζήτησε τον κωδικό του μαθήματος στον πίνακα μαθημάτων
   if ( δεν υπάρχει )
      Γράψε στο log
   else
   {
      Βάλε το (ΑΜ φοιτητή, κωδικός μαθήματος)
                                        στον πίνακα δηλώσεων
      Ενημέρωσε τα sNoOfCourses και sWH του φοιτητή
      Ενημέρωσε το cNoOfStudents του κάθε μαθήματος
   }
} // for
```

Αυτά τα κάνει η:

```
void readCourseCodes( istream& tin, Student& aStudent,
                      PCourse crsArr, int nOfCourses,
                      PStudentInCourse& sic, unsigned int& nOfStdInCrs,
                      unsigned int& resStdInCrs, int sicIncr,
                      ostream& log )
{
   if ( !tin.eof() )
   {
      string str1;
      getline( tin, str1, '\n' );
      if ( !tin.eof() )
      {
         int noc( atoi(str1.c_str()) );
         for ( int k(0); k < noc; ++k )
         {
            if ( nOfStdInCrs+1 == resStdInCrs )
            {
               renew( sic, nOfStdInCrs, resStdInCrs+sicIncr );
               resStdInCrs += sicIncr;
            }
            getline( tin, str1, '\n' );
            if ( !tin.eof() )
            {
               try
               {
                  sic[nOfStdInCrs].setIdNumCCode( aStudent,
                                                  str1.c_str(),
                                                  crsArr,
                                                  nOfCourses );
                  ++nOfStdInCrs;
               }
               catch( StudentInCourseXptn& x )
               {
                  log << " student with id num "
                      << aStudent.getIdNum()
                      << " asking course " << str1 << endl;
               }
            } // if ( !tin.eof()...
         } // for
      }
   }
} // readCourseCodes
```

Prj03.6.5 Η main

Δες πώς (περίπου) θα είναι η **main**:

```
int main()
```

```
{
   PCourse   crsArr;  unsigned int nOfCourses;

   PStudent  stArr;   unsigned int nOfStudents( 0 );
   const int sIncr( 10 );
   unsigned int resStudents;

   PStudentInCourse sic; unsigned int nOfStdInCrs( 0 );
   const int sicIncr( 30 );
   unsigned int resStdInCrs;

   ofstream log;
   try
   {
      loadCourses( "gCourses.dta", crsArr, nOfCourses );

      try { stArr = new Student[sIncr]; }
      catch( bad_alloc )
      {  throw ProgXptn( "main", ProgXptn::allocFailed );  }
      resStudents = sIncr;
      try { sic = new StudentInCourse[sicIncr]; }
      catch( bad_alloc )
      {  throw ProgXptn( "main", ProgXptn::allocFailed );  }
      resStdInCrs = sicIncr;

      log.open( "log.txt" );
      if ( log.fail() )
         throw ProgXptn( "main", ProgXptn::cannotOpen, "log.txt" );
      ifstream tin( "enrllmnt.txt" );
      if ( tin.fail() )
         throw ProgXptn( "main", ProgXptn::cannotOpen, "enrllmnt.txt" );
      while ( !tin.eof() )
      {
         readAStudent( tin, stArr, nOfStudents, resStudents, sIncr );
         if ( !tin.eof() )
         {
            int pos( 0 );
            while ( stArr[pos].getIdNum() !=
                    stArr[nOfStudents-1].getIdNum() ) ++pos;
            if ( pos < nOfStudents-1 )  // υπάρχει ήδη
            {// αγνόησε αυτά που ακολουθούν για τον φοιτητή
               --nOfStudents;
               ignoreStudentData( tin );
               log << " multiple entry for student with id num "
                   << stArr[pos].getIdNum() << endl;
            }
            else  // δεν υπάρχει
            {
               readCourseCodes( tin, stArr[nOfStudents-1],
                                crsArr, nOfCourses,
                                sic, nOfStdInCrs, resStdInCrs,
                                sicIncr, log );
            }
         }
         if ( !tin.eof() )
         {
            string str1;
            getline( tin, str1, '\n' ); // blank line
         }
      } // while
      tin.close();
      for ( int k(0); k < nOfStudents; ++k )
      {
         if ( stArr[k].getWH() > 30 )
            log << "student with id num " << stArr[k].getIdNum()
                << ": " << stArr[k].getWH() << " hours/week"
                << endl;
```

```
      } // for
      save1DTable( "Students.dta", stArr, nOfStudents );
      save1DTable( "enrllmnt.dta", sic, nOfStdInCrs );
      save1DTable( "gCourses.dta", crsArr, nOfCourses );
      delete[] sic;
      delete[] stArr;
      delete[] crsArr;
   } // try
   catch( ProgXptn& x )  { . . . } // catch( ProgXptn
   catch( MyLibXptn& x ) { . . . } // catch( MyLibXptn
   catch( CourseXptn& x ) { . . . } // catch( CourseXptn
   catch( StudentXptn& x ) { . . . } // catch( StudentXptn
   catch( StudentInCourseXptn& x )
   { . . . } // catch( StudentInCourseXptn
   catch( ... ) { . . . }
   log.close();
} // main
```

Παρατηρήσεις ▶

- Ο έλεγχος για την υπέρβαση των 30 ωρών ανά εβδομάδα γίνεται (φυσικά) στο τέλος.
- Η *save1DTable()* υπάρχει στο `MyTmpltLib.h`:

```
// save1DTable -- Φυλάγει σε αρχείο binary με όνομα flNm τα n
//                στοιχεία του πίνακα tbl. Πρέπει να έχει ορισθεί
//                η T::save( ostream& )
template < class T >
void save1DTable( string flNm, T tbl[], int n )
{
   if ( tbl == 0 && n > 0 )
      throw MyTmpltLibXptn( "save1DTable",
                            MyTmpltLibXptn::noArray );
   ofstream bout( flNm.c_str(),
                  ios_base::binary|ios_base::trunc );
   if ( bout.fail() )
      throw MyTmpltLibXptn( "save1DTable",
                            MyTmpltLibXptn::cannotCreate,
                            flNm.c_str() );
   for ( int k(0); k < n; ++k )
   {
      tbl[k].save( bout );
      if ( bout.fail() )
         throw MyTmpltLibXptn( "save1DTable",
                               MyTmpltLibXptn::cannotWrite,
                               flNm.c_str() );
   }
   bout.close();
} // save1DTable
```

- Δηλώσαμε το *log* πριν από την `try`, το ανοίξαμε μέσα στη σύνθετη εντολή της `try` και τον κλείνουμε μετά την τελευταία `catch`. Γιατί; Για να έχουμε τη δυνατότητα να γράφουμε στο *log* σε όποιαν `catch` χρειαστεί.
- Πολλές `catch`! Μια για τις εξαιρέσεις του προγράμματος (*ProgXptn*), μια για τις *MyTmpltLibXptn*, μια για τις εξαιρέσεις της κάθε κλάσης και μια για οτιδήποτε άλλο! ◀

Prj03.7 char* ή *string*;

Τι γίνεται με τα κείμενα (λεκτικά); Θα τα χειριζόμαστε ως `char*` ή ως *string*; Πριν φθάσουμε στο «δια ταύτα» ξαναδιάβασε την §15.13.1 και ας καταγράψουμε μερικά βασικά σημεία.

Γενικώς, *ο τύπος string είναι βολικότερος και τα προγράμματά μας γράφονται πιο εύκολα. Αλλά, το κείμενο αποθηκεύεται σε δυναμική μνήμη.* Συνέπειες σε αυτά που είδαμε μέχρι τώρα:

- Αν στην κλάση εξαιρέσεων δηλώσουμε `string funcName` πρόσεξε τι μπορεί να συμβεί: Προσπαθείς ανεπιτυχώς να πάρεις δυναμική μνήμη και θέλεις να ρίξεις σχετική εξαίρεση. Αφού στην εξαίρεση θα χρειαστεί δυναμική μνήμη το πιο πιθανό είναι να ριχτεί (καινούρια) `std::bad_alloc`. Φυσικά, τέτοια εξαίρεση θα μπορεί να ριχτεί και κατά την (όποια) αντιγραφή της εξαίρεσης και –όπως θα μάθουμε αργότερα– να διακοπεί η εκτέλεση του προγράμματός σου.
- Αν στη *Student* είχαμε δηλώσει

```
string  sSurname;
string  sFirstname;
```

πώς θα κάναμε τη φύλαξη αυτού του μέλους στη *save()*; Θα κάναμε κάτι σαν:

```
char    local[sNameSz];
strncpy( local, sSurname.c_str(), sNameSz-1 );  local[sNameSz-1] = '\0';
bout.write( local, sNameSz );
strncpy( local, sFirstname.c_str(), sNameSz-1 );  local[sNameSz-1] = '\0';
bout.write( local, sNameSz-1 );
```

Δεν σου αρέσει; Στην §15.13.1 λέγαμε: «*Δεν θα μπορούσαμε να φυλάγουμε τις τιμές των ... μελών τύπου string με το ακριβές περιεχόμενο που έχουν κάθε φορά οποιοδήποτε μήκος και αν έχει; Ναι, αρκεί να φυλάγουμε και την τιμή του (κάθε) μήκους.*» Δες λοιπόν και αυτό:

```
unsigned int len( sSurname.length() );
char*        local( new char[len+1] );
strcpy( local, sSurname,c_str() );
bout.write( reinterpret_cast<const char*>(&len), sizeof(len) );
bout.write( local, len+1 );
delete[] local;
len = sFirstname,length();
local = new char[len+1];
strcpy( local, sFirstname );
bout.write( reinterpret_cast<const char*>(&len), sizeof(len) );
bout.write( local, len+1 );
```

Εδώ έχεις αυτό που λέμε «εγγραφές μεταβλητού μήκους».

Είναι φανερό ότι η λύση που έχουμε επιλέξει είναι απλούστερη και από τους δύο τρόπους που είδαμε εδώ.

Φυσικά τίποτε δεν μας εμποδίζει

- να έχουμε τα μέλη των κλάσεων δηλωμένα ως πίνακες `char` αλλά
- να κάνουμε όλους τους άλλους χειρισμούς στον τύπο *string*.

Αυτό κάναμε παραπάνω και έτσι θα συνεχίσουμε. Αλλά, όπως θα δούμε στο επόμενο κεφάλαιο, υπάρχει πιθανότητα –κατ' αρχήν τουλάχιστον– στις αντιγραφές τιμών τύπου *string* να εγερθεί εξαίρεση *bad_alloc*![4]

Prj03.8 Ένα Άλλο Σχέδιο για τις Κλάσεις

Η ιδέα να έχουμε στη *StudentInCourse* μια μέθοδο σαν την

```
void setIdNumCCode( Student& aStudent, string aCCode,
                    Course crsArr[], int nOfCourses );
```

ή, εναλλακτικώς, έναν δημιουργό σαν τον:

```
StudentInCourse::StudentInCourse( Student& aStudent, string aCCode,
                                  Course crsArr[], int nOfCourses )
// . . .
```

δεν είναι και τόσο «ορθόδοξη».

[4] Είπαμε: «*κατ' αρχήν*». Αν είναι να ψάχνουμε για εξαιρέσεις κάθε φορά που κάνουμε κάτι με τιμές τύπου *string*…

Θα έλεγε κανείς ότι καλύτερο θα ήταν να δηλώσουμε τον πίνακα crsArr *μέσα* στη *StudentInCourse*. Δηλαδή θα έχουμε αυτόν τον πίνακακ σε κάθε αντικείμενο *StudentInCourse*; Θα δούμε στα επόμενα μαθήματα ότι υπάρχει τρόπος να το αποφύγουμε.

Πάντως, ούτε αυτό είναι καλό αφού ένας τέτοιος πίνακας πιθανόν να μας χρειάζεται και για άλλες χρήσεις.

Ο συνήθης τρόπος χειρισμού αυτής της κατάστασης είναι να *έχουμε τον πίνακα* crsArr *έξω από την κλάση και μέσα στην κλάση StudentInCourse να έχουμε ένα βέλος προς αυτόν*:

```
class StudentInCourse
{
public:
// . . .
private:
   Course*       crsArr;
   unsigned int* pNOfCourses;
   unsigned int  sicSIdNum;      // αριθμός μητρώου
   char          sicCCode[8];    // κωδικός μαθήματος
}; // StudentInCourse
```

Αλλα το βέλος προς τον πίνακα δεν είναι σταθερό αφού αλλάζει κάθε τόσο με τη *renew()*. Θα το ξαναδούμε αργότερα...

Ερωτήσεις – Ασκήσεις

Α Ομάδα

Prj03-1 Ας πούμε ότι ο κωδικός μαθήματος έχει την εξής δομή:

- Δύο κεφαλαία γράμματα του ελληνικού αλφαβήτου που μπορεί να είναι "ΕΥ" ή "ΤΠ".
- Τέσσερα ψηφία του δεκαδικού συστήματος.
- Ένα κεφαλαίο γράμμα του ελληνικού αλφαβήτου που μπορεί να είναι "Θ" ή "Ε".

Κάνε τις απαραίτητες συμπληρώσεις/διορθώσεις σε αυτά που γράψαμε.