project

1

Αυτοκίνητα στον Δρόμο

Περιεχόμενα:

Prj01.1 Το Πρόβλημα

Θα ξαναλύσουμε το πρόβλημα που λύσαμε στην §13.11 αλλά αυτήν τη φορά με χρήση εξαιρέσεων. Για διευκόλυνσή σου ξαναδίνουμε το πρόβλημα:

> *Ένα συνεργείο έκανε μέτρηση ροής οχημάτων σε κάποιο δρόμο. Σε αρχείο, text – με το όνομα στο δίσκο* **`autoflow.txt`***– καταγράφηκαν οι τιμές ροής σε οχήματα/min κάθε λεπτό. Το πλήθος των τιμών στο αρχείο είναι άγνωστο, αλλά σίγουρα θετικό.*
>
> *Οι πρώτες τρεις γραμμές του αρχείου έχουν την «ταυτότητα» της μέτρησης ως εξής:*

```
Υπεύθυνος:\t<όνομα>\t<επώνυμο>
Σημείο Μετρήσεων:\t<οδός-αριθμός>\t<περιοχή>\t<δήμος>
Αρχή:\tdd.mm.yyyy\thh:mm
```

> *Στις υπόλοιπες γραμμές δίνονται οι τιμές από τη μέτρηση, μια σε κάθε γραμμή. Κάθε τιμή είναι γραμμένη στις πέντε πρώτες θέσεις. Πέρα από τη δέκατη θέση μπορεί να υπάρχουν σχόλια που περιγράφουν συμβάντα κατά τη διάρκεια της μέτρησης. Να ένα παράδειγμα:*

```
Υπεύθυνος:    Ανδρέας       Νικολόπουλος
Σημείο Μετρήσεων: Τζαβέλλα 48    Νεάπολις      Αθήνα
Αρχή:  15.06.2009    05:00
   26
   30
    8
   28
. . .
   17
```

```
   19
    4
   12      / Αρχή λειτουργίας σηματοδότησης
   28
   17
. . .
   17
   31      / Βλάβη σηματοδότη Τζαβέλλα & Γιωργάκου
   23
   24
. . .
```

Να γραφεί πρόγραμμα που θα διαβάζει το αρχείο και θα υπολογίζει:
1. Το πλήθος τιμών του αρχείου καθώς και τη διάρκεια των μετρήσεων σε h και min.
2. Το πλήθος οχημάτων που μετρήθηκαν σε ολόκληρη τη διάρκεια των μετρήσεων.
3. Τη μέση τιμή της ροής οχημάτων κατά τη διάρκεια των μετρήσεων, σε οχήματα/min.
Όλα αυτά θα γράφονται στην τελική έκθεση, που θα γραφεί σε αρχείο με το όνομα `report.txt`. *Στις πρώτες τρεις γραμμές του αρχείου θα αντιγραφούν οι τρεις πρώτες γραμμές του* `autoflow.txt`.
Ένα από τα ζητούμενα της δουλειάς είναι και η μελέτη της μεταβολής της ροής οχημάτων. Ως πρώτο βήμα μας ζητείται να δημιουργήσουμε ένα άλλο αρχείο με τις μεταβολές ροής ανά min.

Prj01.2 Η Δομή Εξαιρέσεων

Στην αρχική λύση μεταφέραμε την πληροφορία «κάτι δεν πάει καλά» με παραμέτρους "`bool& ok`". Τώρα θα τη μεταφέρουμε με εξαιρέσεις. Θα ρίχνουμε εξαιρέσεις τύπου:

```
struct ApplicXptn
{
   enum { . . . };
   char funcName[100];
   int  errCode;
// . . .
   ApplicXptn( const char* fn, int ec, . . . )
   {  strncpy( funcName, fn, 99 );  funcName[99] = '\0';
      errCode = ec;
   // . . .
   }
}; // ApplicXptn
```

όπως τον είδαμε στην §15.10.

Θα συμπληρώνουμε τη δομή όπως θα (ξανα)γράφουμε το πρόγραμμά μας και θα αντιμετωπίζουμε τις ανάγκες του.

Προφανώς θα ξεκινήσουμε από τις συναρτήσεις που έχουν παράμετρο *ok*.

Prj01.3 Η Συνάρτηση *openWrNoReplace()*

Η πρώτη συνάρτηση που θα δούμε είναι η *openWrNoReplace()*. Τη γράφουμε ως:

```
void openWrNoReplace( ofstream& newStream, string flNm )
{
   ifstream test( flNm.c_str() );                        // άνοιξε για διάβασμα
   if ( !test.fail() )                                    // υπάρχει το αρχείο,
   {
      test.close();                            // κλείσε το και μην το πειράξεις
      throw ApplicXptn( "openWrNoReplace", ApplicXptn::fileExists,
```

```
                        flNm.c_str() );
   }
   newStream.open( flNm.c_str() );                          // δημιούργησέ το
   if ( newStream.fail() )
      throw ApplicXptn( "openWrNoReplace", ApplicXptn::cannotCreate,
                        flNm.c_str() );
} // openWrNoReplace
```

Πρόσεξε ότι *η λογική της συνάρτησης είναι ίδια με αυτήν της αρχικής*. Γράφουμε σε σχόλιο «*κλείσε το και μην το πειράξεις*». Αυτό πώς διασφαλίζεται; Με το ότι ακολουθεί εντολή **throw** και εκεί θα διακοπεί η εκτέλεση της συνάρτησης και η "`newStream.open( flNm.c_str() )`" δεν θα εκτελεσθεί!

Εδώ χρησιμοποιούμε δύο κωδικούς σφάλματος που πρέπει να τους ορίσουμε στη δομή εξαιρέσεων:

```
   enum { fileExists, cannotCreate, . . . };
```

Ακόμη προκύπτει ανάγκη για δημιουργό με τρίτη παράμετρο πίνακα χαρακτήρων (ορμαθό της C):

```
   ApplicXptn( const char* fn, int ec, const char* sv="" )
   {  strncpy( funcName, fn, 99 );  funcName[99] = '\0';
      errCode = ec;
      strncpy( errStrVal, sv, 99 );  errStrVal[99] = '\0';  }
```

Το *errStrVal* θα δηλωθεί ως μέλος της δομής:

```
   char errStrVal[100];
```

Prj01.4 Η Συνάρτηση *openFiles()*

Η *openWrNoReplace()* καλείται από την *openFiles()* που –και αυτή– έχει παράμετρο *ok*.

Ξαναδές στην §13.11.1 τη λογική της αρχικής συνάρτησης. Ανοίγουμε τα ρεύματα το ένα μετά το άλλο (*autoflow*, *differences*, *report*) εφόσον δεν υπήρξε πρόβλημα με το άνοιγμα των προηγουμένων. Και εδώ, αν γράψουμε:

```
   autoflow.open( autoFlNm.c_str() );
   if ( autoflow.fail() )
      throw ApplicXptn( "openFiles",
                        ApplicXptn::cannotOpen, autoFlNm.c_str() );
// . . .
```

πετυχαίνουμε το εξής: Αν δεν ανοίξει το *autoflow* ρίχνεται εξαίρεση και δεν προχωρούμε παρακάτω.

Το επόμενο ρεύμα (*differences*) ανοίγεται με την *openWrNoReplace()*. Τώρα, το «δεν άνοιξε το *differences*» σημαίνει ότι ρίχτηκε και μια εξαίρεση. Θα πρέπει να είμαστε έτοιμοι να την πιάσουμε:

```
   try
   {
      openWrNoReplace( differences, difFlNm );
   }
   catch( ApplicXptn& x )
   {
      autoflow.close();
      throw;
   } // catch
```

Τι κάνουμε εδώ; Αν πιάσουμε εξαίρεση κλείνουμε πρώτα το *autoflow* που είναι ήδη ανοικτό και ξαναρίχνουμε την εξαίρεση.

Παρομοίως θα μπορούσαμε να χειριστούμε και το άνοιγμα του *report*:

```
   try
   {
      openWrNoReplace( report, reprtFlNm );
   }
```

```
   catch( ApplicXptn& x )
   {
      autoflow.close();
      differences.close();
      throw;
   } // catch
```

Εδώ φυσικά θα πρέπει να κλείσουμε και το *differences*.

Αφού όμως η εξαίρεση κουβαλάει μαζί της και το όνομα του αρχείου που είχε το πρόβλημα (*x.errStrVal*), μπορούμε να γράψουμε:

```
void openFiles( ifstream& autoflow, string autoFlNm,
                ofstream& differences, string difFlNm,
                ofstream& report, string reprtFlNm )
{
   autoflow.open( autoFlNm.c_str() );
   if ( autoflow.fail() )
      throw ApplicXptn( "openFiles",
                        ApplicXptn::cannotOpen, autoFlNm.c_str() );
// το autoflow ανοικτό
   try
   {
      openWrNoReplace( differences, difFlNm );
      openWrNoReplace( report, reprtFlNm );
   }
   catch( ApplicXptn& x )
   {
      autoflow.close();
      if ( x.errStrVal == reprtFlNm )      // δεν άνοιξε το report
         differences.close();
      throw;
   } // catch
} // openFiles
```

Αν πάρουμε εξαίρεση από την *openWrNoReplace()* κλείνουμε οπωσδήποτε το *autoflow*. Αν προβληματικό αρχείο ήταν το `report.txt` τότε κλείνουμε και το ρεύμα *differences* που έχει ανοίξει πιο πριν.

Για να μην έχουμε τη συνάρτηση «κομμένη στα δύο», δεν θα μπορούσαμε να γράψουμε:

```
   try
   {
      autoflow.open( autoFlNm.c_str() );
      if ( autoflow.fail() )
         throw ApplicXptn( "openFiles",
                           ApplicXptn::cannotOpen,
                           autoFlNm.c_str() );
   // το autoflow ανοικτό
      openWrNoReplace( differences, difFlNm );
      openWrNoReplace( report, reprtFlNm );
   }
   catch( ApplicXptn& x )
   { . . . }
```

Ναι, θα μπορούσαμε. Αλλά προσοχή: αν αποτύχει το άνοιγμα του *autoflow* η εξαίρεση θα συλληφθεί από την **catch** που έχουμε εδώ και η διαχείριση θα είναι πιο πολύπλοκη:

```
   catch( ApplicXptn& x )
   {
      if ( x.errStrVal == difFlNm )   // δεν άνοιξε το differences
         autoflow.close();
      else if ( x.errStrVal == reprtFlNm ) // δεν άνοιξε το report
      {
         autoflow.close();
         differences.close();
      }
      throw;
   } // catch
```

Αν το προτιμάς...

Από τη μετατροπή της *openFiles()* προκύπτει ανάγκη για έναν ακόμη κωδικό σφάλματος:

```
   enum { fileExists, cannotCreate, cannotOpen };
```

Prj01.5 Η Συνάρτηση *copyTitle()*

Στην *copyTitle()* η αντικατάσταση της *ok* με **throw** είναι απλή:

```
void copyTitle( ifstream& autoflow, ofstream& report )
{
   int  lineCount( 0 );   // μετρητής γραμμών
   while ( !autoflow.eof() && lineCount < 3 )
   {
      string aLine;
      getline( autoflow, aLine, '\n' );
      report << aLine << endl;
      ++lineCount;
   } // while (... lineCount < 3)
   if ( lineCount != 3 )
      throw ApplicXptn( "copyTitle", ApplicXptn::incomplete, lineCount );
} // copyTitle
```

Εδώ όμως, πέρα από την ανάγκη για νέο κωδικό σφάλματος (*incomplete*) προκύπτει η ανάγκη και για νέο δημιουργό εξαιρέσεων που να έχει τρίτη παράμετρο τύπου `int`:

```
   ApplicXptn( const char* fn, int ec, int iv )
   {  strncpy( funcName, fn, 99 );  funcName[99] = '\0';
      errCode = ec;   errIntVal = iv;  }
```

Το *errIntVal* είναι μέλος της *ApplicXptn*:

```
   int  errIntVal;
```

Φυσικά, τώρα έχουμε:

```
   enum { fileExists, cannotCreate, cannotOpen, incomplete };
```

Prj01.6 Η Συνάρτηση *closeFiles()*

Η τελευταία συνάρτηση με παράμετρο *ok* είναι η *closeFiles()*. Την ξαναγράφουμε ως εξής:

```
void closeFiles( ifstream& autoflow, ofstream& differences, ofstream& report )
{
   autoflow.close();

   differences.close();
   if ( differences.fail() )
      throw ApplicXptn( "closeFiles", ApplicXptn::cannotClose, "διαφορές" );
   report.close();
   if ( report.fail() )
      throw ApplicXptn( "closeFiles", ApplicXptn::cannotClose, "έκθεση" );
} // closeFiles
```

Αυτή *δεν είναι λειτουργικώς ισοδύναμη με την αρχική*. Αν δεν μπορέσει να κλείσει το *differences* θα ρίξει εξαίρεση και δεν θα προσπαθήσει να κλείσει το *report*. Η αρχική θα προσπαθήσει να κλείσει και τα δύο. Δηλαδή είναι προτιμότερη η αρχική μορφή της συνάρτησης; Ναι, αν έχει νόημα το να πάρουμε το ένα μόνον αρχείο. Αν θέλουμε και τα δύο αρχεία θα πρέπει να ζητήσουμε να ξαναεκτελεσθεί το πρόγραμμα και στις δύο περιπτώσεις.

Πρόσεξε ακόμη ότι επειδή στη συνάρτηση αυτή δεν περνούν τα ονόματα των αρχείων *βάζουμε απλό κείμενο ως τρίτο όρισμα του δημιουργού*.

Prj01.7 Η *ApplicXptn* (τελικώς)

Ας δούμε τώρα πώς έγινε δομή εξαιρέσεων:

```
struct ApplicXptn
{
   enum { fileExists, cannotCreate, cannotOpen, incomplete, cannotClose };
   char funcName[100];
   int  errCode;
   char errStrVal[100];
   int  errIntVal;

   ApplicXptn( const char* fn, int ec, const char* sv="" )
   {  strncpy( funcName, fn, 99 );  funcName[99] = '\0';
      errCode = ec;
      strncpy( errStrVal, sv, 99 );  errStrVal[99] = '\0';  }
   ApplicXptn( const char* fn, int ec, int iv )
   {  strncpy( funcName, fn, 99 );  funcName[99] = '\0';
      errCode = ec;   errIntVal = iv;  }
}; // ApplicXptn
```

Prj01.8 Και η `main`

Τώρα, η `main` θα είναι ως εξής:

```
int main()
{
   ifstream autoflow;    // ρεύμα από το αρχείο autoflow.dta
   ofstream differences; // ρεύμα προς το αρχείο differences.txt
   ofstream report;      // ρεύμα προς το αρχείο report.txt
   string   autoFlNm( "autoflow.txt" ),
            difFlNm( "differences.txt" ),
            reprtFlNm( "report.txt" );
   try
   {
      openFiles( autoflow, autoFlNm, differences, difFlNm,
                 report, reprtFlNm );
      process( autoflow, differences, report );
      closeFiles( autoflow, differences, report );
      cout << "Τέλος καλό, όλα καλά..." << endl;
   }
   catch( ApplicXptn& x )
   {
      switch ( x.errCode )
      {
         case ApplicXptn::fileExists:
            cout << "από την " << x.funcName << ": το αρχείο "
                 << x.errStrVal << " υπάρχει" << endl;
            break;
         case ApplicXptn::cannotCreate:
            cout << "από την " << x.funcName
                 << ": δεν μπορώ να δημιουργήσω το " << x.errStrVal << endl;
            break;
         case ApplicXptn::cannotOpen:
            cout << "από την " << x.funcName
                 << ": δεν μπορώ να ανοίξω το " << x.errStrVal << endl;
            break;
         case ApplicXptn::incomplete:
            cout << "από την " << x.funcName
                 << ": ελλιπή δεδομένα. Διαβάστηκαν "
                 << x.errIntVal << " γραμμές " << endl;
            break;
         case ApplicXptn::cannotClose:
            cout << "από την " << x.funcName
                 << ": δεν μπορώ να κλείσω το αρχείο " << x.errStrVal << endl;
            break;
```

```
         default:
            cout << "unexpected ApplicXptn from " << x.funcName << endl;
      } // switch
   } // catch( ApplicXptn
   catch( ... )
   {
      cout << "unexpected exception" << endl;
   }
} // main
```

Η **main** απλουστεύθηκε σε εντυπωσιακό βαθμό:

```
      openFiles( autoflow, autoFlNm, differences, difFlNm,
                 report, reprtFlNm );
      process( autoflow, differences, report );
      closeFiles( autoflow, differences, report );
      cout << "Τέλος καλό, όλα καλά..." << endl;
```

Αλλα τίποτε δεν είναι δωρεάν στον κόσμο αυτόν: Έφυγαν οι "`if (ok)…`" και μας ήλθε αυτή η μακροσκελής **catch**!

Prj01.9 Η *openFiles()* Αλλιώς

Στην §14.9 γράφαμε: «*Θα διαχειριζόμαστε πάντοτε τις εξαιρέσεις στη* **main***; Όχι. Στη συνέχεια, θα δούμε πώς αποφασίζουμε που και πώς μπορεί να γίνει η καλύτερη διαχείριση της κάθε εξαίρεσης.*» Εδώ διαχειριζόμαστε όλες τις εξαιρέσεις στη **main** αλλά, αν το ξανασκεφτούμε, αυτή δεν είναι η καλύτερη λύση.

Ας πάρουμε τις εξαιρέσεις που ρίχνουμε αν δεν μπορέσουμε να ανοίξουμε κάποιο αρχείο ή κάποια αρχεία· δεν υπάρχει λόγος να τις αφήσουμε να φτάσουν μέχρι τη **main**. Εκεί, αρκεί να φτάσει τον μήνυμα «δεν ανοίγουν τα αρχεία». Καλύτερα λοιπόν να (ξανα)χρησιμοποιήσουμε την *ok* ως εξής:

```
void openFiles( ifstream& autoflow, string autoFlNm,
                ofstream& differences, string difFlNm,
                ofstream& report, string reprtFlNm,
                bool& ok )
{
   ok = false;
   try
   {
      autoflow.open( autoFlNm.c_str() );
      if ( autoflow.fail() )
         throw ApplicXptn( "openFiles", ApplicXptn::cannotOpen,
                           autoFlNm.c_str() );
   // το autoflow ανοικτό
      openWrNoReplace( differences, difFlNm );
      openWrNoReplace( report, reprtFlNm );
      ok = true;
   }
   catch( ApplicXptn& x )
   {
      switch ( x.errCode )
      {
         case ApplicXptn::fileExists:
         case ApplicXptn::cannotCreate:
            autoflow.close();
            if ( x.errStrVal == reprtFlNm )// δεν άνοιξε το report
               differences.close();
            cout << "από την " << x.funcName;
            if ( x.errCode == ApplicXptn::fileExists )
               cout << ": το αρχείο " << x.errStrVal << " υπάρχει" << endl;
            else // x.errCode == ApplicXptn::cannotCreate
               cout << ": δεν μπορώ να δημιουργήσω το " << x.errStrVal
                    << endl;
            break;
```

```
         case ApplicXptn::cannotOpen:
            cout << "από την " << x.funcName
                 << ": δεν μπορώ να ανοίξω το " << x.errStrVal << endl;
            break;
         default:
            throw;
      } // switch
   } // catch
} // openFiles
```

Αυτή η *openFiles()* πιάνει τις εξαιρέσεις που έχουν σχέση με το άνοιγμα των αρχείων: *ApplicXptn* με κωδικούς *fileExists*, *cannotCreate* και *cannotOpen* ενώ αφήνει να περνούν όλες οι άλλες.

Παρατήρηση: ▶

Στην §14.9 γράφαμε ακόμη: «*Οι εξαιρέσεις ρίχνονται μόνο από συναρτήσεις που καλούμε στην ομάδα* ***try****; Όχι! Αργότερα θα δεις παραδείγματα που θα έχουμε εντολές* `throw` *μέσα στην ομάδα της* ***try****.*» Εδώ έχουμε ένα τέτοιο παράδειγμα: περιμένουμε βέβαια εξαιρέσεις από τις δύο κλήσεις στην *openWrNoReplace()* αλλά υπάρχει και **throw** ακριβώς πάνω από αυτές.◀

Η αποτυχία στο άνοιγμα των αρχείων γνωστοποιείται προς τα έξω με τιμή `false` της *ok*. Το τι θα κάνει η **main** σε μια τέτοια περίπτωση είναι άλλη ιστορία. Για παράδειγμα, θα μπορούσε να γίνει κάτι σαν:

```
      do {
         openFiles( autoflow, autoFlNm, differences, difFlNm,
                    report, reprtFlNm, ok );
         if ( !ok )
            getFileNames( autoFlNm, difFlNm, reprtFlNm, quit );
      } while ( !ok && !quit );
      if ( ok )
      {
         process( autoflow, differences, report );
         closeFiles( autoflow, differences, report );
         cout << "Τέλος καλό, όλα καλά..." << endl;
      }
```

Η

```
void getFileNames( string& autoFlNm, string& difFlNm,
                   string& reprtFlNm, bool& quit )
```

(άσκηση για σένα) ζητάει από τον χρήστη να αλλάξει τα ονόματα (κάποιων) αρχείων ή να τα παρατήσει. Στην τελευταία περίπτωση η *quit* επιστρέφει τιμή `true`.

Με τον ίδιο τρόπο μπορούμε να χειριστούμε και την *closeFiles()*.