project

5

Σύνολα Γραμμάτων

Περιεχόμενα:

Prj05.1 Το Πρόβλημα

Το Project αυτό είναι ένα μεγάλο παράδειγμα επιφόρτωσης τελεστών.

Α. Θέλουμε να υλοποιήσουμε έναν τύπο συνόλων που στοιχεία τους θα είναι Κεφαλαία Γράμματα του Λατινικού Αλφαβήτου. Για την υλοποίηση επιλέγουμε τη χρήση ψηφιοχάρτη (bitmap). Δηλαδή ένα σύνολο θα παριστάνεται με τα πρώτα 26 bits μιας τιμής τύπου `long int` *και:*

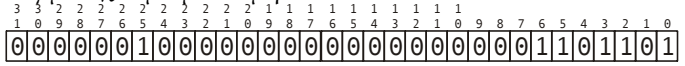
– το bit 0 έχει τιμή 1 αν και μόνον αν το `'A'` *ανήκει στο σύνολο,*

– το bit 1 έχει τιμή 1 αν και μόνον αν το `'B'` *ανήκει στο σύνολο,*

. . .

– το bit 25 έχει τιμή 1 αν και μόνον αν το `'Z'` *ανήκει στο σύνολο,*

Αν για παράδειγμα έχουμε μια τιμή:



αυτή παριστάνει το σύνολο { `'A'`, `'C'`, `'D'`, `'F'`, `'G'`, `'Z'` }. *Προφανώς το* { } *παριστάνεται με την τιμή 0.*

Γράψε μια κλάση, ας την πούμε SetOfUCL,[1] *που θα υλοποιεί το σύνολό μας με τον παραπάνω τρόπο. Η κλάση θα περιέχει μεθόδους για:*

(x, y σύνολα, u άτομο)

– πληθάριθμο: **#***x,*

– εισαγωγή στοιχείου σε σύνολο: x **+=** *u και x* **+** *u,*

– διαγραφή στοιχείου από σύνολο: x **~=** *u και x* **~** *u,*

[1] *Set of U*pper Case *L*etters

– ένωση συνόλων: x ∪= *y και x* ∪ *y,*
– τομή συνόλων: x ∩= *y και x* ∩ *y,*
– διαφορά συνόλων: x \= *y και x* \ *y,*
Φυσικά θα πρέπει να γράψεις και όποιες άλλες μεθόδους ή καθολικές συναρτήσεις θεωρείς αναγκαίες.
Υπόδ.: Για ευκολία μπορείς να υποθέσεις ότι όλα τα Κεφαλαία Γράμματα του Λατινικού Αλφαβήτου βρίσκονται σε συναπτές θέσεις του πίνακα χαρακτήρων κατά αλφαβητική σειρά: αν το `'A'` *βρίσκεται στη θέση p, τότε το* `'B'` *βρίσκεται στη θέση p+1, το* `'C'` *βρίσκεται στη θέση p+2 κ.ο.κ.*
Δίνεται αρχείο text, με όνομα στον δίσκο `mstrpc.txt`*, που περιέχει κείμενο στην αγγλική γλώσσα. Τα γράμματα είναι κεφαλαία και πεζά αλλά για μας είναι ίδια:* `'A'` *και* `'a'` *είναι το ίδιο πράγμα (μια καλή ιδέα είναι η εξής: όταν τα διαβάζουμε, πριν από οποιαδήποτε άλλη δουλειά, τα κάνουμε κεφαλαία). Το κείμενο είναι χωρισμένο σε παραγράφους. Κάθε παράγραφος (εκτός από την τελευταία) χωρίζεται από την επόμενή της με μια κενή γραμμή (δηλαδή: αν διαβάσουμε* `'\n''\n'` *αλλάζει παράγραφος).*
Θέλουμε ένα πρόγραμμα που θα δημιουργεί ένα άλλο αρχείο, μη μορφοποιημένο, με όνομα στο δίσκο `paragltr.dta`*, που για κάθε παράγραφο θα έχει τρεις τιμές κλάσης setOfUCL:*
– το σύνολο των γραμμάτων της παραγράφου (τα βλέπουμε όλα ως κεφαλαία),
– το σύνολο των γραμμάτων της παραγράφου που υπάρχουν και στην επόμενη παράγραφο,
– το σύνολο των γραμμάτων της παραγράφου που δεν υπάρχουν στην επόμενη παράγραφο.

Prj05.2 Παίρνοντας Ιδέες από την Pascal

Πριν προχωρήσουμε στην υλοποίηση της κλάσης, ας ρίξουμε μια ματιά στη διαχείριση συνόλων που επιτρέπει η Pascal που επιτρέπει δηλώσεις της μορφής:

```
x, y, z: set of T;
```

Μετά από αυτό οι *x*, *y*, *z* είναι σύνολα με στοιχεία τύπου *T* (τύπος βάσης). Οι πράξεις συνόλων γίνονται με τους τελεστές:

- "`+`" για την ένωση. Το "`x + y`" παριστάνει το "*x* ∪ *y*"
- "`*`" για την τομή. Το "`x * y`" παριστάνει το "*x* ∩ *y*"
- "`-`" για τη διαφορά. Το "`x - y`" παριστάνει το "*x* \ *y*"

Με τις "`x = y`" και "`x <> y`" συγκρίνονται τα *x*, *y* για ισότητα και ανισότητα αντιστοίχως.

Αν *u* τιμή τύπου *T*, το "`u in x`" είναι συνθήκη που παίρνει τιμή `true` αν η *u ανήκει* στο *x* και `false` αν δεν ανήκει.

Συνθήκη είναι και η "`x <= y`" που παίρνει τιμή `true` αν το *x* είναι *υποσύνολο* του *y* και `false` αν δεν είναι.[2] Η σύγκριση μπορεί να γίνει και με την "`y >= x`" (το *y* είναι *υπερσύνολο* του *x*).

Στην Extended Pascal παρέχεται και η συνάρτηση *card*: Με το "`card(x)`" παίρνουμε τον **πληθάριθμο** (cardinality) του *x* (#*x*).

Δεν είναι κακή ιδέα να επιφορτώσουμε τους αντίστοιχους τελεστές με το ίδιο νόημα για την κλάση που έχουμε να γράψουμε.

---

[2] *Μην* υποθέσεις ότι με το "`x < y`" ελέγχουμε για *γνήσιο υποσύνολο*. Αν τα *x*, *y* είναι σύνολα η Pascal δεν επιτρέπει να γράψεις τέτοια παράσταση.

Prj05.3 Η Κλάση και οι Μέθοδοι

Ξεκινούμε με την αναλλοίωτη της κλάσης η οποία είναι τετριμμένη: `true`. Σύμφωνα με τους κανόνες μας θα πρέπει να γράψουμε `struct` (όλα ανοικτά) και όχι `class`:

```
struct SetOfUCL
{
   unsigned long int bitmap;
// . . .
}; // SetOfUCL
```

Με το πρόβλημα αυτό ασχοληθήκαμε στο Παράδ. 3 της §19.5 όπου λέγαμε «*Αλλά αν επιλέξουμε* `struct` *έχουμε το εξής πρόβλημα: Δίνουμε τη δυνατότητα στο πρόγραμμα που τη χρησιμοποιεί να χειρίζεται την τιμή του (μοναδικού) μέλους ενώ η κλάση γράφεται για να του δώσει τη δυνατότητα να χειρίζεται σύνολα και τα μέλη τους. Πώς αποφεύγεται κάτι τέτοιο;*

- *Με το να την κάνουμε* `class`,
- *να βάλουμε το "*`long int bitmap`*" («μυστικό» της υλοποίησης) σε περιοχή* `private` *και*
- *να μην γράψουμε μεθόδους getBitmap ούτε setBitmap.*»:

```
class SetOfUCL
{
public:
// . . .
private:
   unsigned long int bitmap;
}; // SetOfUCL
```

Όπως λέει και το όνομά της, θα χειριζόμαστε τη *bitmap* ως ψηφιοπίνακα με τις συναρτήσεις (*bitValue(), setBit(), clearBit()* κλπ) που μάθαμε στην §17.6.

Η πιο απλή μορφή της κλάσης, όπως την υποδεικνύει η διατύπωση των απαιτήσεων:

```
class SetOfUCL
{
public:
   SetOfUCL() {  bitmap = 0UL;  };
   ~SetOfUCL() { };
private:
   long int bitmap;
}; // SetOfUCL
```

- Όπως βλέπεις, ορίσαμε και τον ερήμην δημιουργό που μας δίνει το κενό σύνολο. Αν δηλαδή δηλώσουμε:

```
   SetOfUCL x;
// x == { }
```

το *x* είναι κενό.

- Ο «κενός» καταστροφέας μας θυμίζει ότι δεν χρειάζεται να ορίσουμε δημιουργό αντιγραφής, τελεστή εκχώρησης και καταστροφέα μια και αυτοί που θα ορίσει αυτομάτως ο μεταγλωττιστής κάνουν τη δουλειά μας.

Είναι χρήσιμο να έχουμε και έναν δημιουργό (με αρχική τιμή) για δημιουργία *μονοσυνόλων*. Τον δηλώνουμε:

```
   explicit SetOfUCL( char c );
```

και τον γράφουμε αντιγράφοντας από τη *setBit()*.

Σε ποιο δυαδικό ψηφίο θα βάλουμε την τιμή "1"; Αφού το `'A'` αντιστοιχεί στο δυαδικό ψηφίο 0, ένα τυχόν κεφαλαίο γράμμα *c* θα αντιστοιχεί στο δυαδικό ψηφίο:

```
static_cast<int>(c) - static_cast<int>('A')
```

Και αν ο *c* δεν είναι κεφαλαίο γράμμα τί κάνουμε; Τότε *πρέπει να ρίξουμε εξαίρεση* αφού δεν μπορούμε να ανταποκριθούμε:

```
SetOfUCL::SetOfUCL( char c )
{
   if ( !isupper( c ) )
      throw SetOfUCLXptn( "SetOfUCL", SetOfUCLXptn::nonUCL, c );
```

```
   bitmap = 1;
   bitmap <<= ( static_cast<int>(c) - static_cast<int>('A') );
} // SetOfUCL::SetOfUCL
```

Η

```
   SetOfUCL oneSet( 'S' );
```

δημιουργεί ένα (μονο)σύνολο με μοναδικό στοιχείο το `'S'`.

Παρατηρήσεις: ▶

1. Προφανώς θα μπορούσαμε να γράψουμε:

```
   bitmap <<= ( c - 'A' );
```

αλλά προτιμούμε τη γραφή με την τυποθεώρηση για να σου υπενθυμίζουμε τι γίνεται. Είτε με τη μια γραφή είτε με την άλλη, βασιζόμαστε στα εξής:

- *«Για ευκολία μπορείς να υποθέσεις ότι όλα τα Κεφαλαία Γράμματα του Λατινικού Αλφαβήτου βρίσκονται σε συναπτές θέσεις του πίνακα χαρακτήρων κατά αλφαβητική σειρά: αν το* `'A'` *βρίσκεται στη θέση p, τότε το* `'B'` *βρίσκεται στη θέση p+1, το* `'C'` *βρίσκεται στη θέση p+2 κ.ο.κ.»*
- *«Το bit 0 έχει τιμή 1 αν και μόνον αν το* `'A'` *ανήκει στο σύνολο.»*

2. Αν δεν θέλεις να αντιγράψεις τη *setBit()* μπορείς απλώς να την καλέσεις:

```
SetOfUCL::SetOfUCL( char c )
{
   if ( !isupper( c ) )
      throw SetOfUCLXptn( "SetOfUCL", SetOfUCLXptn::nonUCL, c );

   bitmap = 0;
   setBit( bitmap, static_cast<int>(c) - static_cast<int>('A') );
} // SetOfUCL::SetOfUCL
```

Φυσικά, στην περίπτωση αυτή θα πρέπει να φέρεις στο πρόγραμμά σου το περίγραμμα *setBit()*.

Το περίγραμμα της *setBit()* μπορεί να ρίξει και κάποια εξαίρεση· μήπως πρέπει να την πιάσουμε και να ρίξουμε μια *SetOfUCLXptn*; Όχι! Δεν υπάρχει περίπτωση να τη ρίξει! ◀

Μεθόδους "*get*" και "*set*" θα γράψουμε; Για τη *bitmap* ούτε λόγος. Πάντως τον ρόλο "*get*" θα παίξει το κατηγόρημα «*ανήκειν*» (∈) και ρόλο "*set*" θα παίξουν οι μέθοδοι *εισαγωγής στοιχείου σε σύνολο: x* `+=` *u* και *διαγραφής στοιχείου από σύνολο: x* `-=` *u*.

Όπως συνηθίζουμε, για να μπορούμε να κάνουμε τις δοκιμές μας με την κλάση, την εξοπλίζουμε με μια μέθοδο:

```
void SetOfUCL::display( ostream& tout ) const
{
   unsigned long int x( 1 );

   for ( int k(0); k <= 25; ++k )
   {
      if ( (bitmap & x) != 0 )
         tout << static_cast<char>( k + static_cast<int>('A') );
      x <<= 1;
   } // for
   tout << endl;
} // SetOfUCL::display
```

Prj05.3.1 Πληθάριθμος: #*x*

Η πρώτη μέθοδος που μας ζητείται είναι αυτή που δίνει τον *πληθάριθμο* ενός συνόλου *x* (`#`*x*). Ακολουθώντας την Extended Pascal την ονομάζουμε *card*:

x`.`*card*`() == #`*x*

Αφού για κάθε στοιχείο που περιέχεται στο σύνολο θα βάζουμε "1" στο αντίστοιχο δυαδικό ψηφίο του *bitmap*, αρκεί να μετρήσουμε πόσα "1" υπάρχουν στο *bitmap*. Επομένως, μπορούμε να χρησιμοποιήσουμε το περίγραμμα συνάρτησης *count1*:

```
unsigned int SetOfUCL::card() const
{  return count1( bitmap );  } // SetOfUCL::card
```

ή να το αντιγράψουμε με την κατάλληλη προσαρμογή:

```
unsigned int SetOfUCL::card() const
{
   unsigned long int x( 1 );
   int fv( 0 ), lastb( 8*sizeof(unsigned long int)-1 );

   for ( int k(0); k <= lastb; ++k )
   {
      if ( (bitmap & x) != 0 )  ++fv;
      x <<= 1;
   } // for
   return fv;
} // SetOfUCL::card
```

Prj05.3.2Ένωση Συνόλων – Εισαγωγή Στοιχείου σε Σύνολο

Τι σχέση έχει η ένωση συνόλων με την εισαγωγή στοιχείου σε σύνολο; Η εισαγωγή είναι ειδική περίπτωση της ένωσης. Πράγματι, οι προδιαγραφές για την εισαγωγή είναι οι εξής:

true { εισαγωγή του u στο x } $u \in x$

Αλλά τις ίδιες προδιαγραφές έχει και η "$x = x \cup \{ u \}$" ($x \cup= \{ u \}$):

true { $x = x \cup \{ u \}$ } $u \in x$

Αν πάρουμε υπόψη μας ακόμη ότι από την "**operator@=**" μπορούμε να πάρουμε την υλοποίηση της "**operator@**" (επιφόρτωση του "**@**") καταλαβαίνεις ότι το πρώτο που έχουμε να κάνουμε είναι η υλοπόιηση του "$\cup=$". Και –αφού είπαμε ότι θα ακολουθήσουμε τους συμβολισμούς της Pascal– θα επιφορτώσουμε τον "**operator+=**". Όπως είπαμε στην §22.6 η επιφόρτωση θα πρέπει να γίνει με μέθοδο της κλάσης:

```
SetOfUCL& SetOfUCL::operator+=( const SetOfUCL& rhs )
{
   bitmap = bitmap | rhs.bitmap;
   return *this;
} // SetOfUCL::operator+=
```

Αν θέλεις, μπορείς να υλοποιήσεις και μια εκδοχή με όνομα αλλά πρόσεχε: *το "**union**" δεν επιτρέπεται*! Ας πούμε λοιπόν:

```
SetOfUCL& SetOfUCL::setUnion( const SetOfUCL& rhs )
{
   return ( *this += rhs );
} // SetOfUCL::setUnion
```

Πώς λύθηκε το πρόβλημα της εισαγωγής; Αν το x είναι αντικείμενο κλάσης *SetOfUCL* τότε με την

```
   x += SetOfUCL( 'S' );
```

υλοποιείς την $x = x \cup \{$ **'S'** $\}$.

Πώς λύθηκε το πρόβλημα της καθολικής συνάρτησης; Όπως μάθαμε στην §22.7.1:

```
SetOfUCL operator+( const SetOfUCL& lhs, const SetOfUCL& rhs )
{
   SetOfUCL fv( lhs );
   fv += rhs;
   return fv;
} // SetOfUCL operator+
```

Πρόσεξε ότι αυτή η συνάρτηση *δεν χρειάζεται να δηλωθεί ως* **friend**.

Η διατύπωση του προβλήματος όμως απαιτεί να κάνουμε εισαγωγή γράφοντας "`x += 'S'`". Μπορούμε να έχουμε αυτήν τη δυνατότητα; Ναι, με δύο τρόπους:

- Να βγάλουμε το "`explicit`" από τον δημιουργό μονοσυνόλου.
- Να κάνουμε δεύτερη επιφόρτωση του "`+=`".

Αν επιλέξουμε την πρώτη λύση θα είναι δυνατές (μετά την επιφόρτωση του "`==`") σύγκρισεις σαν την "`x == 'S'`" που είναι απαράδεκτη. Έτσι, θα πάμε στη δεύτερη λύση:

```
SetOfUCL& SetOfUCL::operator+=( char c )
{
   if ( !isupper(c) )
      throw SetOfUCLXptn( "operator+=", SetOfUCLXptn::nonUCL, c );

   *this += SetOfUCL( c );
   return *this;
} // SetOfUCL::operator+=
```

Και εδώ μπορούμε να δώσουμε και μια εκδοχή με όνομα:

```
SetOfUCL& SetOfUCL::insert( char c )
{
   return ( *this += c );
} // SetOfUCL::insert
```

Από τη μέθοδο εισαγωγής χαρακτήρα μπορείς να πάρεις –με την πάγια τεχνική– και καθολική συνάρτηση εισαγωγής χαρακτήρα:

```
SetOfUCL operator+( const SetOfUCL& lhs, char rhs )
```

Στα παραπάνω, πρόσεξε το εξής ουσιώδες: *Το νόημα του τελεστή "`+`" δίνεται μόνο μια φορά, στην επιφόρτωση του τελεστή "`+=`"*. Όλοι οι άλλοι ορισμοί ανάγονται αμέσως ή εμμέσως στον "`+=`". Θα πει κανείς: Αν τα ορίσουμε όλα εξ αρχής, γράφοντας πράξεις με ψηφιοχάρτες, θα κάναμε τις συναρτήσεις μας πιο γρήγορες· και μόνο το ότι θα αποφεύγαμε τόσες κλήσεις συναρτήσεων θα ήταν σημαντικό κέρδος. Ναι, αλλά θα είχαμε περισσότερες πιθανότητες για λάθη και η νοηματική συνέπεια όλων αυτών δεν θα ήταν (σχεδόν) αυταπόδεικτη, όπως είναι τώρα.

Παράδειγμα

Οι εντολές

```
   SetOfUCL x( 'A' ), y( x ), z;

   x += 'B';  x.insert( 'C' );
   x.display( cout );
   y += 'C';  y.insert( 'D' );  y += 'E';
   y.display( cout );
   z = x + y;
   z.display( cout );
   x.setUnion( y );
   x.display( cout );
   cout << x.card() << endl;
```

δίνουν:

```
ABC
ACDE
ABCDE
ABCDE
5
```

Prj05.3.3 Διαφορά Συνόλων – Διαγραφή Στοιχείου Συνόλου

Αφου η διαφορά συνόλων $x \setminus y$ είναι το σύνολο των στοιχείων του x που δεν ανήκουν στο y, η διαφορά σχετίζεται με τη διαγραφή στοιχείου συνόλου όπως σχετίζεται η ένωση με την εισαγωγή:

`true` { διαγραφή του u από το x } $u \notin x$

`true` { $x = x \setminus \{ u \}$ } $u \notin x$

Θα πρέπει λοιπόν να ξεκινήσουμε και εδώ με την επιφόρτωση του "`-=`" με μια μέθοδο που θα υλοποιεί την "`\=`". Να σημειώσουμε ότι καταφεύγουμε στον "`-`" που χρησιμοποιεί η Pascal και αγνοούμε τον "`~`" που χρησιμοποιείται στη διατύπωση του προβλήματος διότι ο "`~`" δεν είναι κατάλληλος: είναι *ενικός*!

Μπορούμε να δούμε τη διαφορά $x \setminus y$ ως την τομή, με το x, του συνόλου των στοιχείων που ανήκουν στο x ή στο y (αλλά όχι και στα δύο):

```
SetOfUCL& SetOfUCL::operator-=( const SetOfUCL& rhs )
{
   bitmap = bitmap & ( bitmap ^ rhs.bitmap );
   return *this;
} // SetOfUCL::operator-=
```

και για όποιον προτιμάει συνάρτηση με όνομα:

```
SetOfUCL& SetOfUCL::setDifference( const SetOfUCL& rhs )
{
   return ( *this -= rhs );
} // SetOfUCL::setDifference
```

Από αυτήν παίρνουμε την καθολική συνάρτηση με τη πάγια τεχνική:

```
SetOfUCL operator-( const SetOfUCL& lhs, const SetOfUCL& rhs )
{
   SetOfUCL fv( lhs );
   fv -= rhs;
   return fv;
} // operator-( SetOfUCL, char )
```

και τη μέθοδο διαγραφής:

```
SetOfUCL& SetOfUCL::operator-=( char c )
{
   if ( isupper(c) )
   {
      *this -= SetOfUCL( c );
   }
   return *this;
} // SetOfUCL::operator-=
```

Πρόσεξε ότι εδώ *δεν* ρίχνουμε εξαίρεση αν το c δεν είναι κεφαλαίο λατινικό γράμμα· είναι σίγουρο ότι –και στην περίπτωση αυτή– μετά την εκτέλεση της πράξης $c \notin$ `*this`.

Η μέθοδος

```
SetOfUCL& SetOfUCL::remove( char c )
{
   return ( *this -= c );
} // SetOfUCL::remove
```

κάνει την ίδια δουλειά.

Παράδειγμα

Οι εντολές

```
   x.display( cout );  y.display( cout );
   y -= x;
   y.display( cout );
   y -= 'D';
   y.display( cout );
   y -= 'Q';
   y.display( cout );
   z = x - y;
   z.display( cout );
```

δίνουν:

```
ABC
ACDE
DE
```

```
E
E
ABC
```

Prj05.3.4 Τομή Συνόλων: $x \cap= y$

Ακολουθώντας την Pascal, επιφορτώνουμε τον "`*=`" για την υλοποίηση της πράξης "$\cap=$":

```
SetOfUCL& SetOfUCL::operator*=( const SetOfUCL& rhs )
{
   bitmap = bitmap & rhs.bitmap;
   return *this;
} // SetOfUCL::operator*=
```

Από αυτήν παίρνουμε την καθολική συνάρτηση:

```
SetOfUCL operator*( const SetOfUCL& lhs, const SetOfUCL& rhs )
{
   SetOfUCL fv( lhs );
   fv *= rhs;
   return fv;
} // operator*( SetOfUCL, SetOfUCL )
```

Prj05.4 Το Πρόγραμμα

Ας ξεκινήσουμε με την παρατήρηση ότι: *από κάθε παράγραφο χρειαζόμαστε μόνον το σύνολο των γραμμάτων της* (αφού τα μετατρέψουμε σε κεφαλαία). Έχει νόημα λοιπόν να γράψουμε μια συνάρτηση

```
void readParagraph( ifstream& tin, SetOfUCL& letSet )
```

που θα διαβάζει μέσω του ρεύματος *tin* μια παράγραφο από ένα αρχείο text και θα μας δίνει το σύνολο *letSet* των γραμμάτων που έχει.

Πώς θα δουλεύει η συνάρτηση; Έτσι περίπου:

```
"Άδειασε" το letSet
Διάβασε έναν χαρακτήρα c
while ( δεν τελείωσε η παράγραφος )
{
   if ( ο c είναι γράμμα )
      Βάλε στο letSet το αντίστοιχο κεφαλαίο
   Διάβασε έναν χαρακτήρα c
}
```

Πώς αδειάζουμε το *letSet*; Θα πρέπει να μηδενίσουμε το *bitmap* της *letSet*. Θα πρέπει να εφοδιάσουμε την κλάση μας με μια κατάλληλη μέθοδο[3]:

```
   void clear()  {  bitmap = 0L; };
```

οπότε η "*«Άδειασε» το letSet*" γίνεται:[4]

```
   letSet.clear();
```

Διαβάζουμε τον *c* με την:

```
tin.get(c);
```

Η "*Βάλε στο letSet το αντίστοιχο κεφαλαίο*" γίνεται:

```
letSet.insert( toupper(c) );
```

Ας έλθουμε τώρα στις συνθήκες. Η "*ο c είναι γράμμα*" είναι απλή: `isalpha(c)`.

---

[3] Για να μπορούμε να ελέγχουμε αν ένα σύνολο είναι *κενό* γράφουμε μια:

```
bool isEmpty() const {  return ( bitmap == 0UL );  }
```

[4] Η *clear()* δεν είναι απαραίτητη. Αφού ο ερήμην δημιουργός δημιουργεί ένα κενό σύνολο, μπορούμε να «αδειάσουμε» το *letSet* γράφοντας απλώς: "`letSet = SetOfUCL()`".

Η *"δεν τελείωσε η παράγραφος"* θέλει λίγη σκέψη. Πότε τελειώνει η παράγραφος;

- Όταν βρούμε "`'\n''\n'`" ή
- όταν βρούμε τέλος αρχείου.

Για να μπορούμε να ανιχνεύουμε το "`'\n''\n'`" θα πρέπει να θυμόμαστε τον προηγούμενο χαρακτήρα ή να διαβάζουμε προκαταβολικά (*peek*) τον επόμενο. Ας προτιμήσουμε την πρώτη λύση που δεν αλλάζει και την κανονική ροή της ανάγνωσης. Δηλώνουμε:

```
char c, lastC;
```

και στη *lastC* θα κρατούμε την προηγούμενη τιμή της *c*:

```
lastC = c;  tin.get(c);
```

Έτσι, θα μπορούμε να ελέγξουμε:

```
if ( c == '\n' )
{
   eopar = ( lastC == '\n' );
}
```

Προηγουμένως έχουμε δηλώσει:

```
bool eopar;     // end of paragraph
```

Με βάση τα παραπάνω η *"δεν τελείωσε η παράγραφος"* γίνεται:

```
!tin.eof() && !eopar
```

Αρχικώς βάζουμε:

```
eopar = false;  lastC = '\0';
```

Να λοιπόν η συνάρτηση:

```
void readParagraph( ifstream& tin, SetOfUCL& letSet )
{
   bool eopar;     // end of paragraph
   char c, lastC;

   letSet.clear();
   eopar = false;  lastC = '\0';
   tin.get( c );
   while ( !tin.eof() && !eopar )
   {
      if ( isalpha( c ) )
      {
         letSet.insert( toupper( c ) );
      }
      else if ( c == '\n' )
      {
         eopar = ( lastC == '\n' );
      }
      lastC = c;  tin.get( c );
   } // while
} // readParagraph
```

Τώρα το πρόγραμμά μας γίνεται πιο εύκολο. Ας προσπαθήσουμε να κάνουμε ένα σχέδιο:

```
Διάβασε την πρώτη παράγραφο και πάρε το σύνολο curSet των γραμμάτων της
Διάβασε τη δεύτερη παράγραφο και πάρε το σύνολο nextSet των γραμμάτων της
Υπολόγισε τα σύνολα common = curSet ∩ nextSet και notInNext = curSet \ nextSet
Γράψε στο αρχείο τα curSet, common, notInNext
```

Τώρα θα πρέπει να ξανακάνουμε τα ίδια αλλά το ρόλο της πρώτης παραγράφου θα τον παίζει η δεύτερη και της δεύτερης η τρίτη. Έχουμε όμως μια διαφορά: τη δεύτερη παράγραφο την έχουμε διαβάσει ήδη και τα γράμματά της υπάρχουν στο *nextSet*. Θα δουλέψουμε λοιπόν ως εξής:

```
Βάλε curSet = nextSet
Διάβασε την τρίτη παράγραφο και πάρε το σύνολο nextSet των γραμμάτων της
Υπολόγισε τα σύνολα common = curSet ∩ nextSet και notInNext = curSet \ nextSet
Γράψε στο αρχείο τα curSet, common, notInNext
```

```
Βάλε curSet = nextSet
Διάβασε την τέταρτη παράγραφο και πάρε το σύνολο nextSet των γραμμάτων της
Υπολόγισε τα σύνολα common = curSet ∩ nextSet και notInNext = curSet \ nextSet
Γράψε στο αρχείο τα curSet, common, notInNext
. . .
```

Όταν τελειώσει αυτή η διαδικασία, διότι θα βρούμε τέλος αρχείου, θα έχουμε στο *curSet* το σύνολο των γραμμάτων της τελευταίας παραγράφου. Τι θα κάνουμε με αυτό; Το πιο φυσιολογικό είναι να θεωρήσουμε ότι υπάρχει και μια «κενή παράγραφος» (με κενό σύνολο γραμμάτων) μετά από αυτήν.

Να λοιπόν τι θα πρέπει να κάνουμε:

```
Διάβασε την πρώτη παράγραφο και πάρε το σύνολο curSet των γραμμάτων της
while ( !tin.eof() )
{
   Διάβασε την επόμενη παράγραφο και πάρε το σύνολο nextSet των
   γραμμάτων της
   Υπολόγισε τα σύνολα
   common = curSet ∩ nextSet και notInNext = curSet \ nextSet
   Γράψε στο αρχείο τα curSet, common, notInNext
   Βάλε curSet = nextSet
}
Βάλε nextSet = { }
Υπολόγισε τα σύνολα common = curSet ∩ nextSet και notInNext = curSet \ nextSet
Γράψε στο αρχείο τα curSet, common, notInNext
```

Ας γράψουμε το πρόγραμμά μας. Στην αρχή δηλώνουμε:

```
   SetOfUCL curSet, nextSet, common, notInNext;
   ifstream tin( "mstrpc.txt" );
   ofstream bout( "paragltr.dta", ios::binary );
```

και αρχίζουμε τη μετάφραση:

Η «*Διάβασε την πρώτη παράγραφο και πάρε το σύνολο curSet των γραμμάτων της*» μεταφράζεται στην:

```
   readParagraph( tin, curSet );
```

Παρομοίως η «*Διάβασε την επόμενη παράγραφο και πάρε το σύνολο nextSet των γραμμάτων της*» γίνεται:

```
   readParagraph( tin, nextSet );
```

Η «*Υπολόγισε τα σύνολα common = curSet ∩ nextSet και notInNext = curSet \ nextSet*« γίνεται:

```
      common = curSet * nextSet;
      notInNext = curSet - nextSet;
```

Και με την «*Γράψε στο αρχείο τα curSet, common, notInNext*» τι κάνουμε;

Ένας τρόπος να την υλοποιήσουμε είναι να γράψουμε:

```
      bout.write( reinterpret_cast<char*>( &curSet ),
                  sizeof( curSet ) );
```

και στη συνέχεια τα παρόμοια για τα άλλα δύο σύνολα.

Ένας άλλος είναι να εφοδιάσουμε την κλάση μας με μια μέθοδο:

```
void SetOfUCL::save( ostream& bout ) const
{
   bout.write( reinterpret_cast<const char*>(&bitmap), sizeof(bitmap) );
} // SetOfUCL::save
```

και να έχουμε το πρόγραμμά μας πιο απλό:

```
      curSet.save( bout );
      common.save( bout );  notInNext.save( bout );
```

Να λοιπόν το πρόγραμμά μας:

```
#include <fstream>
#include <iostream>
#include <cctype>
```

```
#include "SetOfUCL.cpp"

using namespace std;

void readParagraph( ifstream& tin, SetOfUCL& letSet );

int main()
{
   SetOfUCL curSet, nextSet, common, notInNext;
   ifstream tin( "mstrpc.txt" );
   ofstream bout( "paragltr.dta", ios::binary );

   readParagraph( tin, curSet );
   while ( !tin.eof() )
   {
      readParagraph( tin, nextSet );
      common = curSet * nextSet;
      notInNext = curSet - nextSet;
      curSet.save( bout );
      common.save( bout );  notInNext.save( bout );
      curSet = nextSet;
   } // while
   tin.close();
   nextSet.clear();
   common = curSet * nextSet;
   notInNext = curSet - nextSet;
   curSet.save( bout );
   common.save( bout );  notInNext.save( bout );
   bout.close();
} // main

void readParagraph( ifstream& tin, SetOfUCL& letSet )
// ΟΠΩΣ ΠΑΡΑΠΑΝΩ
```

Prj05.4.1 Και η *SetOfUCL* Μέχρι Τώρα

Λύσαμε το πρόβλημά μας έχοντας την εξής κλάση:

```
class SetOfUCL
{
public:
   SetOfUCL() {  bitmap = 0UL;  };
   explicit SetOfUCL( char c );
   ~SetOfUCL() { };
   unsigned int card() const;
   SetOfUCL& operator+=( const SetOfUCL& rhs );
   SetOfUCL& setUnion( const SetOfUCL& rhs );
   SetOfUCL& operator+=( char c );
   SetOfUCL& insert( char c );
   SetOfUCL& operator-=( const SetOfUCL& rhs );
   SetOfUCL& setDifference( const SetOfUCL& rhs );
   SetOfUCL& operator-=( char c );
   SetOfUCL& remove( char c );
   SetOfUCL& operator*=( const SetOfUCL& rhs );
   SetOfUCL& setIntersection( const SetOfUCL& rhs );
   void clear()  {  bitmap = 0UL; };
   bool isEmpty() const  {  return ( bitmap == 0UL );  };
   void save( ostream& bout ) const;
private:
   unsigned long int bitmap;
}; // SetOfUCL

SetOfUCL operator+( const SetOfUCL& lhs, const SetOfUCL& rhs );
SetOfUCL operator+( const SetOfUCL& lhs, char rhs );
SetOfUCL operator-( const SetOfUCL& lhs, const SetOfUCL& rhs );
```

```
SetOfUCL operator-( const SetOfUCL& lhs, char rhs );
SetOfUCL operator*( const SetOfUCL& lhs, const SetOfUCL& rhs );
```

Prj05.5 Εμπλουτίζοντας την Κλάση

Συγκρίνοντας τα παραπάνω με αυτά που ξέρουμε από τη Θεωρία Συνόλων βλέπουμε ότι μας λείπουν δύο πάγιες διμελείς σχέσεις των συνόλων: η σχέση "*ανήκειν*":[5]

$\_ \in \_ : X \leftrightarrow \mathbb{P}\, X$

και η σχέση "*περιέχεσθαι*":

$\_ \subseteq \_ : \mathbb{P}\, X \leftrightarrow \mathbb{P}\, X$

όπου:

$\forall x, y: \mathbb{P}\, X \bullet (x \subseteq y \Leftrightarrow \forall u{:}X \bullet u \in x \Rightarrow u \in y)$

Ας ξεκινήσουμε από την "$\in$". Υπάρχει κάποιος τελεστής για να επιφορτώσουμε; Η C++ δεν έχει "`in`" και δεν φαίνεται να ταιριάζει κάποιος άλλος τελεστής. Θα πρέπει να χρησιμοποιήσουμε κάποιο όνομα, π.χ.:

"u `isMemberOf` x" ή "x `hasMember` u"

Αν θέλουμε να γράψουμε μια μέθοδο καλύτερα να χρησιμοποιήσουμε το δεύτερο αφού για την "$u \in x$" θα γράφουμε "`x.hasMember(u)`".

Θα υλοποιήσουμε τη μέθοδο που θα επιστρέφει τιμή τύπου *bool* –αφού είναι κατηγόρημα– και θα γράψουμε προσαρμόζοντας την *bitValue()*:

```
bool SetOfUCL::hasMember( char c ) const
{
   bool fv( false );

   if ( isupper( c ) )
   {
      unsigned long int x( 1UL );
      x <<= static_cast<int>(c) - static_cast<int>('A');
      fv = ( ( bitmap & x ) != 0 );
   }
   return fv;
} // SetOfUCL::hasMember
```

Πρόσεξε ότι όταν ο *c* δεν είναι κεφαλαίο γράμμα δεν ρίχνουμε εξαίρεση· λέμε απλώς ότι δεν ανήκει στο σύνολο.

Αν προτιμάς το πρώτο όνομα, καλύτερα γράψε μια καθολική συνάρτηση:

```
bool isMemberOf( char c, const SetOfUCL& rhs )
```

που θα την καλείς ως εξής: "`isMemberOf( u, x )`". Φυσικά, αυτήν θα πρέπει να δηλώσεις ως **friend** της κλάσης.[6]

Για την υλοποίηση της δεύτερης δεν μας διευκολύνει και τόσο το κατηγόρημα που δίνουμε παραπάνω. Η συνάρτηση που θα προκύψει θα είναι πολύ αργή. Ένας άλλος τρόπος είναι να στηριχθούμε στην ιδιότητα:

$$(x \subseteq y) \Leftrightarrow (x = x \cap y)$$

Έχουμε λοιπόν:

```
bool SetOfUCL::isSubset( SetOfUCL rhs ) const
{  return ( bitmap == ( bitmap & rhs.bitmap ) );  } // isSubset
```

Αν *x*, *y* είναι *SetOfUCL* τότε η συνθήκη "$x \subseteq y$" θα «μεταφράζεται» στο πρόγραμμά μας σε "`x.isSubset(y)`".

---

[5] Θυμίσου ότι κάθε σύνολο με στοιχεία από το (*σύνολο βάσης*) *X* είναι μέλος του *δυναμοσυνόλου* $\mathbb{P}\, X$ του *X*.

[6] Βέβαια, αν δεν φοβάσαι μπερδέματα, μπορείς να έχεις και τις δύο, αρκεί να ορίσεις:

```
bool isMemberOf( char c, const SetOfUCL& rhs )  {  return rhs.hasMember( c );  }
```

Η Pascal χρησιμοποιεί τον "<=" για το υποσύνολο. Μπορούμε λοιπόν να τον επιφορτώσουμε ως καθολική συνάρτηση:

```
bool operator<=( const SetOfUCL& lhs, const SetOfUCL& rhs )
{
   return lhs.isSubset( rhs );
} // operator<=( SetOfUCL...
```

και αντί για "x.isSubset(y)" να γράφουμε "x <= y".

Αφού η Pascal χρησιμοποιεί και τον ">=" για το υπερσύνολο, μπορούμε να τον επιφορτώσουμε χρησιμοποιώντας την *isSubset()*:

```
bool operator>=( const SetOfUCL& lhs, const SetOfUCL& rhs )
{
   return rhs.isSubset( lhs );
} // operator<=( SetOfUCL...
```

Δεν πρέπει να παραλείψουμε να επιφορτώσουμε τον "==" για να μπορούμε να συγκρίνουμε δύο σύνολα για ισότητα:

```
bool operator==( const SetOfUCL& lhs, const SetOfUCL& rhs )
{
   return ( lhs.bitmap == rhs.bitmap );
} // operator==( const SetOfUCL
```

και

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

Πρόσεξε ότι η πρώτη θα πρέπει να δηλωθεί στην κλάση ως **friend**.[7]

Αν θέλουμε να έχουμε τη δυνατότητα να ελέγχουμε για γνήσιο *υποσύνολο* (proper subset) γράφουμε:

```
bool SetOfUCL::isProperSubset( const SetOfUCL& rhs ) const
{
   return ( this->isSubset(rhs) && (bitmap != rhs.bitmap) );
} // isProperSubset
```

και επιφορτώνουμε:

```
bool operator<( const SetOfUCL& lhs, const SetOfUCL& rhs )
{
   return lhs.isProperSubset( rhs );
} // SetOfUCL::operator<
```

Παρατήρηση: ▶
Εδώ προσοχή: *Η "x < y" και η "x >= y" δεν έχουν την παραμικρή σχέση!* Το ότι το *x* δεν είναι γνήσιο υποσύνολο του *y* –δηλαδή δεν ισχύει η "x < y"– δεν σημαίνει ότι το *x* είναι υπερσύνολο του *y*. Μάλλον η ιδέα να επιφορτώσουμε τον "<" δεν είναι και τόσο καλή! ◀

Τελικώς, το περιεχόμενο του SetOfUCL.h έχει γίνει:

```
#ifndef _SETOFUCL_H
#define _SETOFUCL_H

#include <string>
#include <fstream>

using namespace std;

class SetOfUCL
{
friend bool operator==( const SetOfUCL& lhs, const SetOfUCL& rhs );
public:
   SetOfUCL() {  bitmap = 0UL;  };
```

[7] Να γράψουμε "return (lhs.isSubset(rhs) && rhs.isSubset(lhs))" για να αποφύγουμε το **friend**; Ε, όχι και έτσι...

```
   explicit SetOfUCL( char c );
   ~SetOfUCL() { };
   unsigned int card() const;
   SetOfUCL& operator+=( const SetOfUCL& rhs );
   SetOfUCL& setUnion( const SetOfUCL& rhs );
   SetOfUCL& operator+=( char c );
   SetOfUCL& insert( char c );
   SetOfUCL& operator-=( const SetOfUCL& rhs );
   SetOfUCL& setDifference( const SetOfUCL& rhs );
   SetOfUCL& operator-=( char c );
   SetOfUCL& remove( char c );
   SetOfUCL& operator*=( const SetOfUCL& rhs );
   SetOfUCL& setIntersection( const SetOfUCL& rhs );
   void clear()  {  bitmap = 0UL; };
   bool isEmpty() const  {  return ( bitmap == 0UL );  };
   void save( ostream& bout ) const;

   bool hasMember( char c ) const;
   bool isSubset( SetOfUCL rhs ) const;
   bool isProperSubset( const SetOfUCL& rhs ) const;

   void display( ostream& tout ) const;
private:
   unsigned long int bitmap;
}; // SetOfUCL

SetOfUCL operator+( const SetOfUCL& lhs, const SetOfUCL& rhs );
SetOfUCL operator+( const SetOfUCL& lhs, char rhs );
SetOfUCL operator-( const SetOfUCL& lhs, const SetOfUCL& rhs );
SetOfUCL operator-( const SetOfUCL& lhs, char rhs );
SetOfUCL operator*( const SetOfUCL& lhs, const SetOfUCL& rhs );

bool operator<=( const SetOfUCL& lhs, const SetOfUCL& rhs );
bool operator<( const SetOfUCL& lhs, const SetOfUCL& rhs );
bool operator!=( const SetOfUCL& lhs, const SetOfUCL& rhs );

struct SetOfUCLXptn
{
   enum { nonUCL };
   char funcName[100];
   int  errorCode;
   char errCharVal;
   SetOfUCLXptn( const char* mn, int ec, char c=0 )
      :  errorCode( ec ),  errCharVal( c )
   {  strncpy( funcName, mn, 99 );  funcName[99] = '\0';  }
}; // SetOfUCLXptn
```

Prj05.6 Σχόλια και Παρατηρήσεις

Ας επαναλάβουμε και εδώ μερικές παρατηρήσεις σχετικώς με τις διάφορες επιλογές που κάναμε αναπτύσσοντας την κλάση.

- Για την ένωση και τη διαφορά ορίσαμε (επιφορτώσαμε) τους "`+=`" και "`-=`" και από αυτούς πήραμε όλους τους άλλους τελεστές (και σχετικές μεθόδους) σχεδόν με «μηχανικό τρόπο». Έτσι έχουμε εξασφαλισμένο ότι ο "`+`" έχει παντού το ίδιο νόημα (το ίδιο ισχύει και για τον "`-`"). Αν κάτι πρέπει να αλλάξει, αυτό θα γίνει σε ένα σημείο μόνο.
- Η τεχνική που μάθαμε στην §22.7.1 και εφαρμόσαμε για να πάμε από τον "`+=`" στον "`+`" και από τον "`-=`" στον "`-`" έχει έναν περιορισμό: δεν μπορεί να επιστρέψει τύπο αναφοράς. Πρόσεξε την περίπτωση με τη διαφορά:

```
SetOfUCL operator-( const SetOfUCL& lhs, const SetOfUCL& rhs )
{
   SetOfUCL fv( lhs );
   fv -= rhs;
```

```
   return fv;
} // operator-( SetOfUCL, SetOfUCL )
```

Όταν κληθεί η `operator-()` δημιουργείται η *fv* που

- είναι τοπική στη συνάρτηση και
- ζει όσο εκτελείται η συνάρτηση.

Όταν έλθει η στιγμή να χρησιμοποιηθεί η τιμή που επιστρέφει –για παράδειγμα εκτελεσθεί η εκχώρηση στη *notInNext* στην "`notInNext = curSet - nextSet`"– η εκτέλεση της `operator-()` έχει τελειώσει, η *fv* δεν υπάρχει πια και η συνάρτηση επιστρέφει ένα βέλος προς μια ανύπαρκτη μεταβλητή.

Τι θα μπορούσαμε να κάνουμε; Να κάνουμε την *fv* δυναμική και να γράψουμε τη συνάρτηση έτσι ώστε να επιστρέφει βέλος προς αυτήν.

- Κατά τα άλλα η τεχνική αυτή δεν είναι μόνο για τελεστές. Ας πούμε ότι έχουμε την

```
SetOfUCL& SetOfUCL::setDifference( const SetOfUCL& rhs )
{
   return ( *this -= rhs );
} // SetOfUCL::setDifference
```

και θέλουμε να ορίσουμε και την καθολική συνάρτηση *setDifference()* που θα κάνει την ίδια δουλειά με την `operator-()`. Γράφουμε:

```
SetOfUCL setDifference( const SetOfUCL& lhs, const SetOfUCL& rhs )
{
   SetOfUCL fv( lhs );
   fv.setDifference( rhs );
   return fv;
} // setDifference( SetOfUCL, SetOfUCL )
```