ΕΚΔΟΣΕΙΣ ΚΛΕΙΔΑΡΙΘΜΟΣ

C++ για Μαθηματικούς

Μια εισαγωγή για σπουδαστές και καθηγητές

Edward Scheinerman

Περιεχει δωρεάν CD-ROM

Επιστημονική επιμέλεια ελληνικής έκδοσης:
Γιώργος Στεφανίδης
Τμήμα Εφαρμοσμένης Πληροφορικής, Πανεπ. Μακεδονίας

Περιεχόμενα

Κεφάλαιο 7

Πυθαγόρειες τριάδες

Πυθαγόρεια τριάδα είναι μια λίστα τριών ακεραίων (a, b, c) τέτοιων, ώστε $a^2 + b^2 = c^2$. Μια τέτοια τριάδα λέγεται *πρωτεύουσα* αν οι ακέραιοι είναι μη αρνητικοί και gcd(a, b, c) = 1. Παραδείγματος χάριν, η (3, 4, 5) είναι μια Πυθαγόρεια τριάδα επειδή 32 + 42 = 52. Αν και οι (30, 40, 50) και (– 3, 4, –5) είναι επίσης Πυθαγόρειες τριάδες, δεν είναι πρωτεύουσες.

Σε αυτό και το επόμενο κεφάλαιο αναπτύσσουμε το μηχανισμό της C++ που είναι απαραίτητος για να βρούμε όλες τις πρωτεύουσες Πυθαγόρειες τριάδες με a, b, $c \le 100$ (ή οποιαδήποτε άλλη δεδομένη τιμή).

7.1 Παραγωγή Πυθαγόρειων τριάδων

Υπάρχει μια απλή μέθοδος παραγωγής Πυθαγόρειων τριάδων. Έστω z ένας ακέραιος του Gauss· δηλαδή, $z = m + ni$ όπου $m, n \in \mathbb{Z}$. Θεωρούμε το $|z^4|$. Είναι

$$\left|z^4\right| = \left(\left|z\right|^2\right)^2 = \left(m^2 + n^2\right)^2$$

αλλά

$$\left|z^4\right| = \left|z^2\right|^2 = \left|\left(m^2 - n^2\right) + \left(2mn\right)i\right|^2 = \left(m^2 - n^2\right)^2 + \left(2mn\right)^2.$$

Επομένως

$$\left(m^2 - n^2\right)^2 + \left(2mn\right)^2 = \left(m^2 + n^2\right)^2. \qquad (*)$$

Επειδή $m, n \in \mathbb{Z}$, έπεται ότι το $(m^2 - n^2, 2mn, m^2 + n^2)$ είναι μια Πυθαγόρεια τριάδα. Αυτό μπορεί επίσης να ελεγχθεί απευθείας με ανάπτυξη των δύο μελών της ταυτότητας (*).

Παρότι κάθε τριάδα της μορφής $(m^2 - n^2, 2mn, m^2 + n^2)$ είναι μια Πυθαγόρεια τριάδα, δεν προκύπτουν όλες οι Πυθαγόρειες τριάδες με τον τρόπο αυτό.

Παραδείγματος χάριν, η (9, 12, 15) δεν μπορεί να εκφραστεί έτσι. [*Απόδειξη*. Αν η (9, 12, 15) ήταν αυτής της μορφής, τότε $2mn = 12$, οπότε $mn = 6$. Αυτό συνεπάγεται ότι $\{m, n\} = \{\pm 2, \pm 3\}$ ή $\{\pm 1, \pm 6\}$, κανένα από τα οποία δεν δίνει (9, 12, 15).]

Μπορούμε όμως να δείξουμε ότι κάθε *πρωτεύουσα* Πυθαγόρεια τριάδα είναι της μορφής $(m^2 - n^2, 2mn, m^2 + n^2)$. Χρησιμοποιούμε το γεγονός αυτό για να δημιουργήσουμε μια κλάση η οποία αντιπροσωπεύει πρωτεύουσες Πυθαγόρειες τριάδες.

7.2 Σχεδίαση μιας κλάσης πρωτευουσών Πυθαγόρειων τριάδων

Θέλουμε να σχεδιάσουμε μια κλάση της C++ που να αντιπροσωπεύει τις πρωτεύουσες Πυθαγόρειες τριάδες. Η πρώτη ερώτηση είναι: Πως θα ονομάσουμε αυτή την κλάση; Το πλέον περιγραφικό όνομα είναι το φλύαρο `PrimitivePythagoreanTriple`, αλλά θα ήταν επώδυνο να το πληκτρολογούμε επανειλημμένα. Στο άλλο άκρο, θα μπορούσαμε να ονομάσουμε την κλάση `PPT`, αλλά θα αποκρύπταμε κάθε πληροφορία για το περιεχόμενό της. Καταλήγουμε σε μια συμβιβαστική λύση: `PTriple`.

Πώς θα διατηρεί μια `PTriple` τα δεδομένα της; Η ευκολότερη λύση είναι να αποθηκεύσουμε την τριάδα (a, b, c). Αν και θα μπορούσαμε να παραλείψουμε το c επειδή μπορεί να υπολογιστεί από τα a και b, δεν αποτελούν πρόβλημα τα λίγα bit μνήμης που χρησιμοποιούνται για την αποθήκευση του c. Δεν επιθυμούμε να θεωρούμε την (3, 4, 5) διαφορετική από την οποιαδήποτε μετάθεση της τριάδας (3, 4, 5), οπότε ας συμφωνήσουμε ότι μια `PTriple` θα κρατά τους τρεις ακεραίους σε μη φθίνουσα σειρά. Οι τρεις ακέραιοι αποθηκεύονται σε μεταβλητές τύπου `long` με ονόματα a, b και c.

Κάθε κλάση χρειάζεται μια κατασκευάστρια μέθοδο. Σύμφωνα με όσα είπαμε, η κατασκευάστρια μέθοδος θα πρέπει να επεξεργάζεται ένα ζεύγος ακεραίων (m, n) για να παράγει την τριάδα $(m^2 - n^2, 2mn, m^2 + n^2)$. Δυστυχώς, η ζωή δεν είναι τόσο απλή όσο η ανάθεση

$$a \leftarrow m^2 - n^2, \quad b \leftarrow 2mn, \qquad c \leftarrow m^2 + n^2.$$

Κατ' αρχήν, κάτι τέτοιο θα είχε ως ενδεχόμενο αποτέλεσμα το a ή το b να είναι αρνητικό. Ακόμα κι αν είναι και τα δύο θετικά, μπορεί να μην είναι στη σωστή σειρά (θέλουμε $0 \le a \le b$). Τέλος, οι τρεις τιμές μπορεί να μην είναι σχετικώς

πρώτοι αριθμοί. Το μόνο για το οποίο μπορούμε να είμαστε βέβαιοι είναι ότι το $m^2 + n^2$ είναι μη αρνητικός ακέραιος και ο μεγαλύτερος από τους τρεις.

Για να αντιμετωπίσει αυτά τα ζητήματα, η κατασκευάστρια μέθοδος χρειάζεται να διορθώσει τα πρόσημα των *a* και *b*, να τους βάλει στη σωστή σειρά, και να διαιρέσει με το μέγιστο κοινό διαιρέτη gcd(*a*, *b*).

Και κάτι τελευταίο. Σε περίπτωση που ισχύει $m = n = 0$, η τριάδα που κατασκευάζεται θα είναι η (0, 0, 0). Τεχνικά, αυτή δεν είναι πρωτεύουσα επειδή οι καταχωρίσεις δεν είναι σχετικώς πρώτοι αριθμοί. Έχουμε δύο επιλογές: μπορούμε να επιτρέψουμε η (0, 0, 0) να είναι μια έγκυρη `PTriple`, ή μπορούμε να το απαγορεύσουμε. Και στις δύο περιπτώσεις, η κατασκευάστρια μέθοδος πρέπει να χειριστεί αυτή την πιθανότητα ως μια ειδική περίπτωση. Στο βιβλίο αυτό αποφασίζουμε να θεωρήσουμε την (0, 0, 0) ως μια έγκυρη `PTriple`.

Χρειάζεται επίσης να έχουμε μια κατασκευάστρια μέθοδο χωρίς ορίσματα. Είναι αυτή που καλείται μέσω μιας δήλωσης της μορφής `PTriple P;` Τι είδους Πυθαγόρεια τριάδα θα πρέπει να δημιουργεί αυτή; Με άλλα λόγια, ποια θα είναι η προκαθορισμένη πρωτεύουσα Πυθαγόρεια τριάδα; Υπάρχουν τρεις φυσικές επιλογές: (0, 0, 0), (0, 1, 1), ή (3, 4, 5). Η τελευταία είναι η μικρότερη τριάδα στην οποία κανένα από τα στοιχεία δεν είναι μηδέν. Καλώς ή κακώς, αποφασίζουμε η προκαθορισμένη να είναι η (0, 0, 0). Αν δεν είστε ικανοποιημένοι με οποιαδήποτε απ' αυτές τις επιλογές μπορείτε ασφαλώς να επιλέξετε κάτι διαφορετικό.

Τι μεθόδους χρειαζόμαστε; Χρειάζεται να ξέρουμε τις τιμές που είναι αποθηκευμένες στις *a*, *b* και *c*, οπότε προσανατολιζόμαστε για μεθόδους `getA()`, `getB()`, και `getC()` που θα αναφέρουν αυτές τις τιμές. Δεν θέλουμε μεθόδους οι οποίες να μπορούν να τροποποιούν αυτές τις τιμές, επειδή δεν θέλουμε να επιτρέψουμε στο χρήστη να θέσει μια `PTriple` σε μια μη έγκυρη κατάσταση. Αν είχαμε μια μέθοδο `setA`, τότε η `P.setA(2);` θα είχε ως αποτέλεσμα μια `PTriple` η οποία θα μπορούσε να μην είναι μια Πυθαγόρεια τριάδα.

Θέλουμε επίσης τελεστές που να συγκρίνουν τριάδες `PTriple`. Εκτός από τους τελεστές == και `!=`, ορίζουμε έναν τελεστή <. Χρειαζόμαστε έναν τελεστή μικρότερο από, έτσι ώστε να μπορούμε να ταξινομούμε τριάδες `PTriple` σε μια συστοιχία και να διαγράφουμε διπλότυπα. (Να θυμόμαστε ότι ο σκοπός μας είναι να βρούμε όλες τις πρωτεύουσες Πυθαγόρειες τριάδες με $0 \le a \le b \le c \le 1000$.) Έχουμε κάποια επιλογή για το πώς να ορίσουμε τον τελεστή < για τριάδες `PTriple`· χρειάζεται απλώς να εξασφαλίσουμε ότι ο ορισμός του τελεστή < οδηγεί σε μια πλήρη διάταξη. Η απλή λύση μας είναι να διατάξουμε τις τριάδες λεξικογραφικά. Θέτουμε $(a, b, c) < (a', b', c')$ αν, $c < c'$, ή διαφορετικά,

αν $c = c'$ και $b < b'$. Φυσικά, αν $c = c'$ και $b = b'$, τότε $a = a'$ και οι τριάδες ταυτίζονται.

Τέλος είναι χρήσιμο να ορίσουμε τον τελεστή << έτσι ώστε να μπορούμε να εμφανίζουμε τριάδες `PTriple` στην οθόνη του υπολογιστή.

Είμαστε τώρα σε θέση να υλοποιήσουμε την κλάση PTriple.

7.3 Υλοποίηση της κλάσης `PTriple`

Όπως σχεδιάστηκε, η κλάση `PTriple` έχει τρία στοιχεία δεδομένων τύπου `long` με ονόματα a, b, και c. Οι κατασκευάστριες μέθοδοι εξασφαλίζουν ότι ισχύουν πάντα οι ακόλουθες συνθήκες.

$$0 \le a \le b \le c, \quad \gcd(a, b) = 1, \quad a^2 + b^2 = c^2. \qquad (*)$$

Η μόνη εξαίρεση είναι ότι επιτρέπουμε να ισχύει $(a, b, c) = (0, 0, 0)$.

Δημιουργούμε την κλάση ώστε να έχει δύο κατασκευάστριες μεθόδους: μία που δεν δέχεται ορίσματα και μία που δέχεται δύο ορίσματα τύπου `long`. Η πρώτη κατασκευάζει την (0, 0, 0) και η δεύτερη χρησιμοποιεί τη μέθοδο που περιγράψαμε στην αρχή του κεφαλαίου. Είναι αρμοδιότητα της κατασκευάστριας μεθόδου να διασφαλίσει ότι ισχύουν οι συνθήκες της (∗). Αυτό το παραπέμπουμε σε μια βοηθητική μέθοδο η οποία ονομάζεται `reduce`. (Κυριολεκτώντας, δε χρειαζόμαστε μια ξεχωριστή διαδικασία γι' αυτά τα βήματα, όμως θέλουμε να δώσουμε την εικόνα ενός στιγμιότυπου μιας ιδιωτικής μεθόδου για μια κλάση.)

Οι δημόσιες μέθοδοι της `PTriple` είναι οι `getA`, `getB`, `getC` και οι τελεστές ==, != και <. Τέλος, έχουμε μια διαδικασία `operator<<` η οποία δεν είναι μέλος της κλάσης `PTriple`.

Παρουσιάζουμε τώρα το αρχείο κεφαλίδας `PTriple.h`. Σε αυτή την περίπτωση, δεν έχουμε διαγράψει τα σχόλια. Αυτό το αρχείο κεφαλίδας εισάγει κάποιες νέες ιδέες με τις οποίες θα ασχοληθούμε μετά την παρουσίαση του αρχείου.

Πρόγραμμα 7.1: Αρχείο κεφαλίδας για την κλάση `PTriple`.

```
1  #ifndef PTRIPLE_H
2  #define PTRIPLE_H
3  #include <iostream>
4  using namespace std;
5
```

```
/**
 * Μια PTriple αντιπροσωπεύει μια ανηγμένη Πυθαγόρεια τριάδα. Δηλαδή μια
 * ακολουθία τριών μη αρνητικών ακεραίων (a,b,c) σε μη φθίνουσα
 * διάταξη, τέτοια ώστε a^2+b^2=c^2 και οι (a,b,c) είναι σχετικώς
 * πρώτοι. Η απαίτηση "σχετικώς πρώτοι" σημαίνει ότι ασχολούμαστε μόνο
 * με πρωτεύουσες Πυθαγόρειες τριάδες. Επιτρέπουμε την τριάδα (0,0,0).
 */
class PTriple {
private:
  /// Η μικρότερη πλευρά της τριάδας
  long a;
  /// Η μεγαλύτερη πλευρά της τριάδας
  long b;
  /// Η υποτείνουσα
  long c;

  /**
   * Αυτή η ιδιωτική μέθοδος διασφαλίζει ότι τα στοιχεία της τριάδας
   * είναι μη αρνητικοί, σε μη φθίνουσα διάταξη, και σχετικώς πρώτοι.
   */
  void reduce();

public:
  /**
   * Προκαθορισμένη κατασκευάστρια μέθοδος. Δημιουργεί την τριάδα (0,0,0).
   */
  PTriple() {
    a = b = c = 0;
  }

  /**
   * Κατασκευή από ένα ζεύγος ακεραίων. Δοθέντων των ακεραίων m και n,
   * δημιουργούμε μια Πυθαγόρεια τριάδα παίρνοντας τις πλευρές να είναι 2mn
   * και m^2-n^2 και την υποτείνουσα να είναι m^2+n^2. Μετά
   * σιγουρευόμαστε ότι οι τρεις αριθμοί είναι μη αρνητικοί, σε μη
   * φθίνουσα διάταξη, και μετά διαιρούμε με τον ΜΚΔ τους.
   * Π.χ., η PTriple(2,1) δημιουργεί τη γνωστή τριάδα (3,4,5).
   */
  PTriple(long m, long n);

  /// Ποια είναι η μικρότερη πλευρά αυτής της τριάδας;
  long getA() const {
    return a;
  }

  /// Ποια είναι η μεγαλύτερη πλευρά αυτής της τριάδας;
  long getB() const {
```

```
53      return b;
54    }
55
56    /// Ποια είναι η υποτείνουσα αυτής της τριάδας;
57    long getC() const {
58      return c;
59    }
60
61    /**
62     * Ελέγχουμε αν αυτή η τριάδα PTriple είναι μικρότερη από μια άλλη. Η διάταξη
63     * είναι λεξικογραφική αρχίζοντας με το c, και μετά το b. Δηλαδή πρώτα
64     * συγκρίνουμε υποτείνουσες. Αν είναι ίσες συγκρίνουμε τη μεγαλύτερη πλευρά.
65     * @param that Μια άλλη PTriple
66     * @return true αν αυτή η PTriple είναι λεξικογραφικά μικρότερη
67     */
68    bool operator<(const PTriple& that) const;
69
70    /**
71     * Ελέγχουμε αν αυτή η PTriple είναι ίση με μια άλλη.
72     */
73    bool operator==(const PTriple& that) const {
74      return ( (a==that.a) && (b==that.b) && (c==that.c) );
75    }
76
77    /**
78     * Ελέγχουμε αν αυτή η PTriple δεν είναι ίση με μια άλλη.
79     */
80    bool operator!=(const PTriple& that) const {
81      return !( (*this) == that );
82    }
83  };
84
85  /// Στέλνουμε μια PTriple σε ένα ρεύμα εξόδου
86  ostream& operator<<(ostream& os, const PTriple& PT);
87
88  #endif
```

Η δήλωση κλάσης αρχίζει στη γραμμή 13. Εντός της ενότητας `private` βλέπουμε τις τρεις μεταβλητές τύπου `long` που κρατούν τα δεδομένα της τριάδας. Έχοντας στο μυαλό μας ένα ορθογώνιο τρίγωνο με μήκη πλευρών *a*, *b*, *c*, αποκαλούμε το *a* μικρότερη πλευρά, το *b* μεγαλύτερη πλευρά και το *c* υποτείνουσα.

Μέσα επίσης στην ενότητα `private` υπάρχει η μέθοδος `reduce()`. Αυτή είναι μια μέθοδος που καλείται από την κατασκευάστρια μέθοδο των δύο ορισμάτων `PTriple(m,n)`· θα αναφερθούμε σ' αυτή με περισσότερες λεπτομέρειες παρακάτω.

Η ενότητα `public` αρχίζει στη γραμμή 28 με τις δύο κατασκευάστριες μεθόδους. Η άνευ ορίσματος κατασκευάστρια μέθοδος `PTriple()` δεν δηλώνεται όπως θα περιμέναμε. Κανονικά, μια δήλωση διαδικασίας δεν περιλαμβάνει τον κώδικα της διαδικασίας. Όλο κι όλο που περιμένουμε είναι αυτό:

```
PTriple();
```

Αντί γι' αυτό, βλέπουμε εκείνα που θα περιμέναμε κανονικά να βρούμε στο αρχείο `.cc` .

```
PTriple() {
  a = b = c = 0;
}
```

Πρόκειται για ένα ιδιαίτερο χαρακτηριστικό της C++. Μπορούν να ορίζονται διαδικασίες σε αρχεία κεφαλίδας, και αυτές είναι γνωστές ως *ενδογραμμικές* (inline) διαδικασίες. Η ενδογραμμική διαδικασία συνδυάζει τη δήλωση και τον ορισμό μιας διαδικασίας σε μία μόνο θέση στο αρχείο κεφαλίδας.

Μπορούμε να μετατρέψουμε οποιαδήποτε διαδικασία σε ενδογραμμική διαδικασία. Η δεσμευμένη λέξη `inline` χρησιμοποιείται για να ανακοινώσει αυτό το γεγονός. Για διαδικασίες όμως που είναι μέλη κλάσεων (δηλαδή μεθόδους) η δεσμευμένη λέξη είναι προαιρετική. Για όλες τις άλλες διαδικασίες (δηλαδή συνηθισμένες μεθόδους που δεν είναι μέλη κλάσεων) η δεσμευμένη λέξη `inline` είναι υποχρεωτική.

Πότε θα πρέπει να χρησιμοποιείτε ενδογραμμικές διαδικασίες, αντί της συνήθους τεχνικής μιας δήλωσης στο αρχείο `.h` και ενός ορισμού στο αρχείο `.cc`; Αν ο κώδικας της μεθόδου είναι μόνο μία γραμμή ή δύο, τότε είναι καλή ιδέα να τη γράφετε ως ενδογραμμική διαδικασία. Αρκετές από τις μεθόδους της `PTriple` είναι ιδιαίτερα σύντομες και τις γράφουμε όντως ως ενδογραμμικές διαδικασίες. Άλλες (όπως η κατασκευάστρια μέθοδος με τα δύο ορίσματα ή η `operator<`) είναι πιο περίπλοκες, οπότε γι' αυτές χρησιμοποιούμε τη συνηθισμένη τεχνική.

Υπάρχουν τα υπέρ και τα κατά στο να γράφουμε ενδογραμμικό κώδικα. Είναι βολικό να γράφουμε τον κώδικα για μια μέθοδο στο ίδιο αρχείο όπου ορίζεται. Ο μεταγλωττιστής της C++ παράγει αποδοτικότερο κώδικα για διαδικασίες που είναι γραμμένες ενδογραμμικά. (Πάντως, οι έξυπνοι μεταγλωττιστές

της C++ μπορούν να καταλάβουν πότε αξίζει τον κόπο να μετατρέπουν μια συνηθισμένη διαδικασία σε ενδογραμμική διαδικασία.) Όμως, ο αντικειμενικός κώδικας για προγράμματα που χρησιμοποιούν ενδογραμμικές διαδικασίες πιάνει κατά τι περισσότερο χώρο στο δίσκο. Επίσης, παίρνει λίγο περισσότερο χρόνο η μεταγλώττιση προγραμμάτων με ενδογραμμικές διαδικασίες.

Μερικές φορές είναι υποχρεωτικό να γράψουμε μεθόδους ως ενδογραμμικές διαδικασίες. Όταν έρθει η κατάλληλη στιγμή θα το εξηγήσουμε.

Οι διαδικασίες που δεν καθορίζονται από ενδογραμμικό κώδικα στο αρχείο κεφαλίδας ορίζονται στο αρχείο `PTriple.cc` που παρουσιάζουμε στη συνέχεια.

Πρόγραμμα 7.2: Αρχείο προγράμματος για την κλάση `PTriple`.

```
1  #include "PTriple.h"
2  #include "gcd.h"
3
4  PTriple::PTriple(long m, long n) {
5    a = 2*m*n;
6    b = m*m - n*n;
7    c = m*m + n*n;
8
9    reduce();
10 }
11
12 void PTriple::reduce() {
13   // Δεν γίνεται τίποτα αν a=b=c=0
14   if ((a==0) && (b==0) && (c==0)) return;
15
16   // Διασφαλίζουμε ότι a,b είναι μη αρνητικοί (ο c πρέπει να είναι)
17   if (a<0) a = -a;
18   if (b<0) b = -b;
19
20   // Διασφαλίζουμε ότι a <= b
21   if (a>b) {
22     long tmp = a;
23     a = b;
24     b = tmp;
25   }
26
27   // Διασφαλίζουμε ότι οι a,b,c είναι σχετικώς πρώτοι
28   long d = gcd(a,b);
29   a /= d;
30   b /= d;
```

```
31    c /= d;
32  }
33
34  bool PTriple::operator<(const PTriple& that) const {
35    if (c < that.c) return true;
36    if (c > that.c) return false;
37
38    if (b < that.b) return true;
39    return false;
40  }
41
42  ostream& operator<<(ostream& os, const PTriple& PT) {
43    os << "(" << PT.getA() << "," << PT.getB() << ","
44       << PT.getC() << ")";
45    return os;
46  }
```

Ο κώδικας για την κατασκευάστρια μέθοδο `PTriple(m,n)` διασπάται σε δύο μέρη. Δοθέντων των ακεραίων εισόδου *m*, *n* θέτουμε

$$a \leftarrow 2mn, \qquad b \leftarrow m^2 - n^2, \quad \text{και} \qquad c \leftarrow m^2 + n^2.$$

Μετά χρειάζεται να ρυθμίσουμε ορισμένα πράγματα. Χρειάζεται να σιγουρέψουμε ότι οι *a* και *b* είναι μη αρνητικοί, ότι $a \le b$, και ότι $\gcd(a, b, c) = 1$. Αυτές τις δουλειές τις παραπέμπουμε στην ιδιωτική μέθοδο `reduce` (η οποία καλείται από την κατασκευάστρια μέθοδο `PTriple` στη γραμμή 9).

Ακολουθεί ο κώδικας για τον τελεστή <. Ο τελεστής αυτός χρησιμοποιείται για να συγκρίνει μια δεδομένη `PTriple` με μια άλλη. Αν *R* και *P* είναι μεταβλητές τύπου `PTriple`, η παράσταση `R<P` έχει ως αποτέλεσμα να εκτελεστεί η διαδικασία `operator<` που ανήκει στο `R`.

Πρώτα συγκρίνουμε τις τιμές του `c` στα `R` και `P`· αυτό συμβαίνει στις γραμμές 35–36. Η πρώτη σύγκριση είναι `c < that.c`. Το πρώτο `c` είναι η υποτείνουσα του ορίσματος του αριστερού μέλους του < (δηλαδή `R`). Το "ακέφαλο" `c` αναφέρεται στο `c` του αντικειμένου επί του οποίου κλήθηκε η μέθοδος. Το `c` του ορίσματος του δεξιού μέλους χρειάζεται να καθοριστεί, οπότε πρέπει να αναφερόμαστε σ' αυτό ως `that.c` επειδή το όρισμα του δεξιού μέλους ονομάζεται `that`. Επειδή το `P` μεταβιβάζεται κατ' αναφορά στον τελεστή < (απαιτείται για τους τελεστές), η `that.c` είναι ακριβώς η ίδια μεταβλητή με την `P.c`.

Εν συντομία: όταν εμπλέκεται η `R<P`, το "ακέφαλο" `c` στη γραμμή 35 είναι το μέλος δεδομένων `c` του `R` και το `that.c` στη γραμμή 35 είναι το μέλος δεδομένων `c` του `P`.

7.4 Εύρεση και ταξινόμηση των τριάδων

Ο σκοπός μας είναι να βρούμε όλες τις Πυθαγόρειες τριάδες (a, b, c) με $0 \le a \le b \le c \le 1000$. Η στρατηγική μας είναι η εξής.

- Πρώτον, δημιουργούμε μια συστοιχία τριάδων `PTriple` καλώντας την `PTriple(m,n)` για ένα κατάλληλα μεγάλο διάστημα τιμών των `m` και `n`. Επειδή κάθε πρωτεύουσα Πυθαγόρεια τριάδα μπορεί να προκύψει από την $(m^2 - n^2, 2mn, m^2 + n^2)$ δεν χρειάζεται να θεωρήσουμε τιμές του m ή του n μεγαλύτερες του $\sqrt{1000}$.

  Το άνω φράγμα 1000 είναι φυσικά αυθαίρετο. Η σχεδίασή μας επιτρέπει οποιοδήποτε άνω φράγμα N.

- Καθώς παράγονται οι Πυθαγόρειες τριάδες, τις αποθηκεύουμε σε μια συστοιχία. Επειδή το άνω φράγμα των a, b, c είναι αυθαίρετο (το N καθορίζεται από το χρήστη), η συστοιχία που δημιουργούμε χρειάζεται να δεσμεύεται δυναμικά.

  Πόσο μεγάλη θα πρέπει να είναι αυτή η συστοιχία; Στο σενάριο χειρότερης περίπτωσης, όλες οι Πυθαγόρειες τριάδες που παράγουμε μπορεί να είναι πρωτεύουσες και διαφορετικές η μία με την άλλη. (Αυτή είναι στην πραγματικότητα μια χονδροειδής υπερεκτίμηση, αλλά εξυπηρετεί τους σκοπούς μας.) Επειδή κάνουμε επαναλήψεις επί όλων των m και n με $1 \le m, n \le \sqrt{N}$, μια συστοιχία που μπορεί να φιλοξενήσει N τιμές είναι αρκετά μεγάλη.

- Άπαξ και συμπληρωθεί η συστοιχία, την ταξινομούμε. Στην εκ των προτέρων ανάγκη της ταξινόμησης οφείλεται το ότι ορίσαμε τον τελεστή <.

- Τελικώς, διαβάζουμε όλη τη συστοιχία εμφανίζοντας στην οθόνη τα μοναδικά στοιχεία.

Παρουσιάζουμε το πρόγραμμα που εκτελεί όλα αυτά τα βήματα· μετά την παρουσίαση του κώδικα εξηγούμε τα κύρια χαρακτηριστικά του.

Πρόγραμμα 7.3: Ένα πρόγραμμα για την εύρεση Πυθαγόρειων τριάδων.

```
#include "PTriple.h"
#include <iostream>
#include <cmath>
using namespace std;
/**
 * Βρίσκει όλες τις πρωτεύουσες Πυθαγόρειες τριάδες (a,b,c) με
 * 0 <= a<= b<= c<= N όπου το N καθορίζεται από το χρήστη.
 */

int main() {
  PTriple* table;    // πίνακας για να αποθήκευση των τριάδων
  long N;            // άνω φράγμα στις τριάδες.

  // Ζητά από το χρήστη το N
  cout << "Please enter N (upper bound on triples) --> ";
  cin >> N;
  if (N <= 0) return 0; // δεν κάνει τίποτα όταν το N δεν είναι θετικό

  // Δεσμεύει χώρο για τον πίνακα
  table = new PTriple[N];

  // Συμπληρώνει τον πίνακα με όλες τις πιθανές τριάδες PTriple
  long idx = 0;        // δείκτης μέσα στον πίνακα
  long rootN = long( sqrt(double(N)) );

  for (long m=1; m<=rootN; m++) {
    for (long n=1; n<=rootN; n++) {
      PTriple P = PTriple(m,n);
      if (P.getC() <= N) {
        table[idx] = P;
        idx++;
      }
    }
  }

  // Ταξινομεί τον πίνακα
  sort(table, table+idx);

  // Εμφανίζει στην οθόνη (όχι διπλότυπα) στοιχεία του πίνακα
  cout << table[0] << endl;
  for (int k=1; k<idx; k++) {
    if (table[k] != table[k-1]) {
      cout << table[k] << endl;
    }
  }

```

```
47    // Ελευθερώνει μνήμη που καταλαμβανόταν από τον πίνακα
48    delete[] table;
49    return 0;
50  }
```

Σημείωση: Ορισμένοι μεταγλωττιστές μπορεί να απαιτούν την οδηγία `#include <algorithm>` προκειμένου να χρησιμοποιήσουν τη διαδικασία `sort` (γραμμή 37).

Για την ανάλυση τώρα του κώδικα.

- Χρειαζόμαστε έναν πίνακα για τις τριάδες `PTriple`, οπότε δηλώνουμε μια μεταβλητή `table` τύπου `PTriple*` (δείκτης σε ένα `PTriple`, δηλαδή την κεφαλή μιας συστοιχίας). Επειδή δεν γνωρίζουμε πόσο μεγάλη χρειάζεται να είναι αυτή η συστοιχία, δεν μπορούμε να τη δηλώσουμε με μια εντολή της μορφής `PTriple table[100];`.
- Οι γραμμές 15–17 ζητούν από το χρήστη να μας δώσει έναν ακέραιο N. Αν ο χρήστης μας δώσει έναν ακέραιο ο οποίος είναι μικρότερος του 1, τερματίζουμε το πρόγραμμα.
- Η γραμμή 20 δεσμεύει χώρο για τη συστοιχία `table`.
- Η μεταβλητή `idx`, που δηλώνεται στη γραμμή 23, χρησιμοποιείται για προσπέλαση στοιχείων της συστοιχίας `table`.
- Η μεταβλητή `rootN` ορίζεται ίση με $\lfloor \sqrt{N} \rfloor$. Η σύνταξη, δυστυχώς, είναι άβολη. Η διαδικασία `sqrt` της C++ (η οποία μπορεί να απαιτεί την οδηγία `#include <cmath>`) δέχεται και επιστρέφει τιμές τύπου `double`. Έτσι, χρειάζεται πρώτα να μετατρέψουμε ρητά (cast) την τιμή της N σε `double`, να υπολογίσουμε την τετραγωνική της ρίζα, και μετά να τη μετατρέψουμε πίσω σε τύπο `long`. Όταν μια μεταβλητή τύπου `double` μετατρέπεται σε τύπο `long`, τα ψηφία δεξιά της υποδιαστολής παραβλέπονται. Επομένως, η γραμμή 24 υπολογίζει το επιθυμητό.
- Οι γραμμές 26–34 διατρέχουν όλες τις δυνατές τιμές των m, n με $1 \le m, n \le \sqrt{N}$. Για κάθε ζεύγος, παράγουμε μια πρωτεύουσα Πυθαγόρεια τριάδα. Αν η c-τιμή της δεν είναι μεγαλύτερη του N (γραμμή 29) εισάγουμε την τιμή της στον πίνακα και αυξάνουμε κατά 1 την `idx`. Επομένως, αφού τελειώσουμε αυτόν το διπλό βρόχο `for`, η μεταβλητή `idx` έχει το πλήθος των καταχωρίσεων που κάναμε στη συστοιχία `table`.

- Η ταξινόμηση του πίνακα λαμβάνει χώρα στη γραμμή 37. Η διαδικασία `sort` της C++ μπορεί να χρησιμοποιηθεί σε οποιαδήποτε συστοιχία στοιχείων, αρκεί ο τύπος αυτών των στοιχείων να επιδέχεται σύγκριση με έναν τελεστή <. Έτσι, η `sort` μπορεί να χρησιμοποιηθεί για την ταξινόμηση μιας συστοιχίας ακεραίων τύπου `long` ή `double`. Για να ταξινομήσουμε μια συστοιχία τριάδων `PTriple`, χρειάζεται απλώς να εξασφαλίσουμε έναν τελεστή <.

  Η διαδικασία `sort` μπορεί να απαιτεί την κεφαλίδα `algorithm`.

  Η επίκληση της ταξινόμησης γίνεται με μια ανορθόδοξη σύνταξη: `sort(table, table+idx);`. Το πρώτο όρισμα, `table`, έχει νόημα. Η διαδικασία `sort` χρειάζεται να γνωρίζει τι συστοιχία έχει να ταξινομήσει. Το δεύτερο όρισμα είναι που δυσκολευόμαστε να καταλάβουμε.

  Το δεύτερο όρισμα στη `sort` χρειάζεται να είναι ένας δείκτης προς τη θέση του πρώτου στοιχείου *πέρα από το τέλος της συστοιχίας*. Με άλλα λόγια, αν η συστοιχία έχει πέντε στοιχεία (δεικτοδοτημένα από 0 έως 4), τότε το δεύτερο όρισμα στη `sort` χρειάζεται να είναι ένας δείκτης προς το ανύπαρκτο *έκτο* στοιχείο της συστοιχίας (αριθμοδείκτης 5).

  Γενικότερα, η `sort` μπορεί να χρησιμοποιηθεί για την ταξινόμηση ενός συνεχόμενου μπλοκ στοιχείων μιας συστοιχίας. Το πρώτο όρισμα στη `sort` θα πρέπει να είναι ένας δείκτης στην αρχή του μπλοκ και το δεύτερο όρισμα θα πρέπει να είναι ένας δείκτης *μετά* το τέλος του μπλοκ.

  Ξέρουμε ότι το `table` είναι ένας δείκτης προς το πρώτο στοιχείο της συστοιχίας `table`. Δηλαδή, το `table` έχει τη διεύθυνση μνήμης του `table[0]`. Στη C++, το `table+1` παίρνει ως τιμή τη διεύθυνση μνήμης που έχει το `table[1]`. Έτσι το `table+idx` είναι η θέση μνήμης που έχει το `table[idx]`. Όμως, σε αυτό το σημείο του προγράμματός μας, το `idx` έχει έναν αριθμό μεγαλύτερο από τη θέση όπου τοποθετήσαμε τελευταία μια `PTriple`. Αυτό είναι που θέλει η `sort` και έτσι αυτό είναι που κάνουμε.

  Τελευταία γραμμή: Για την ταξινόμηση μιας συστοιχίας, με όνομα έστω `table`, η οποία περιέχει έστω 25 στοιχεία, η εντολή που χρησιμοποιούμε είναι αυτή: `sort(table, table+25);`. Δεν υπάρχει κανένα πρόβλημα να ξεχάσουμε ό,τι είπαμε γύρω από τους δείκτες και απλώς να θυμόμαστε το εξής:

  ```
  sort(table_name, table_name + number_of_elements);
  ```

- Οι γραμμές 40–45 διατρέχουν τα στοιχεία της `table`. Αν το τρέχον στοιχείο είναι διαφορετικό από το προηγούμενο, το εμφανίζουμε στην οθόνη.
- Στο τέλος, στη γραμμή 48, ελευθερώνουμε τη μνήμη που έχει δεσμευτεί για την `table`. Μιλώντας αυστηρά, αυτό δεν είναι αναγκαίο επειδή βρισκόμαστε στο τέλος του προγράμματος και θα γινόταν αυτόματα η ανάκτηση της μνήμης. Παρόλα αυτά, είναι μια καλή συνήθεια να διασφαλίζουμε ότι κάθε `new` εντολή αντιστοιχίζεται με μια εντολή `delete[]`.

Αυτό που βλέπουμε όταν εκτελείται το πρόγραμμα (για $N = 100$) είναι το εξής.

```
Please enter N (upper bound on triples) --> 100
(0,1,1)
(3,4,5)
(5,12,13)
(8,15,17)
(7,24,25)
(20,21,29)
(12,35,37)
(9,40,41)
(28,45,53)
(11,60,61)
(33,56,65)
(16,63,65)
(48,55,73)
(36,77,85)
(13,84,85)
(39,80,89)
(65,72,97)
```

Βλέπουμε ότι υπάρχουν 17 πρωτεύουσες Πυθαγόρειες τριάδες με $0 \le a \le b \le c \le 100$. Αυτό συνεπάγεται ότι η συστοιχία που δημιουργήσαμε, η `table`, χρησιμοποίησε περισσότερη μνήμη απ' ότι στην πραγματικότητα χρειαζόμασταν.

Ας πούμε και τις ενστάσεις μας για αυτό το πρόγραμμα.

- Σπαταλάει μνήμη. Αυτό δεν αποτελεί τρομερό πρόβλημα επειδή η αποθήκευση εκατοντάδων, ή ακόμα και εκατοντάδων χιλιάδων Πυθαγόρειων τριάδων, είναι στις δυνατότητες ακόμα και των πιο μέτριων υπολογιστών. Ωστόσο, μπορεί να συναντήσουμε άλλες καταστάσεις στις οποίες χρειάζεται να είμαστε προσεκτικοί ως προς την ποσότητα μνήμης που χρησιμοποιούμε.

- Ήταν ενοχλητικό ότι χρειαζόταν να υπολογίζουμε πόση μνήμη πρέπει να θέτουμε κατά μέρος στην `table`. Θα ήταν πολύ ευκολότερο αν ο πίνακας μπορούσε να προσαρμόζει το μέγεθός του για να ικανοποιεί τις ανάγκες μας, αντί να απαιτείται από μας να υπολογίζουμε πόσο μεγάλο να τον κάνουμε.
- Η κλήση της `sort` με μπερδεύει ακόμα. Η προσθήκη ενός αντικειμένου τύπου `PTriple*` και ενός αντικειμένου τύπου `long` φαίνεται απλώς εσφαλμένη. (Δεν είναι λάθος, αλλά προκαλεί σύγχυση.)

Θα ασχοληθούμε με αυτές τις ενστάσεις στο επόμενο κεφάλαιο.

7.5 Ασκήσεις

7.1 Δημιουργήστε μια κλάση `Interval` που αντιπροσωπεύει κλειστά διαστήματα στην ευθεία των πραγματικών αριθμών: $[a, b] = \{x \in \mathbb{R} : a \leq x \leq b\}$. Υλοποιήστε πλήρως αυτή την κλάση σε ένα αρχείο `Interval.h` χωρίς κανένα αρχείο κώδικα. Για να το πετύχετε αυτό θα χρειαστεί να κάνετε όλες τις μεθόδους και τις διαδικασίες ενδογραμμικές.

Η κλάση θα πρέπει να περιλαμβάνει τα ακόλουθα χαρακτηριστικά.

(α) Δύο κατασκευάστριες μεθόδους: μία κατασκευάστρια μέθοδο χωρίς όρισμα η οποία κατασκευάζει ένα προκαθορισμένο διάστημα (έστω το [0, 1]) και μία κατασκευάστρια μέθοδο δύο ορισμάτων που κατασκευάζει το διάστημα με τα καθορισμένα άκρα. Ως απόκριση είτε στην `Interval(3,4)` είτε στην `Interval(4,3)` θα πρέπει να κατασκευάζεται το διάστημα [3,4].

(β) Μέθοδοι get για να αποκαλύπτουν τα άκρα του διαστήματος.

(γ) Τελεστές σύγκρισης για ισότητα (==) και ανισότητα (`!=`).

(δ) Μια διαδικασία `operator<` για λεξικογραφική ταξινόμηση των διαστημάτων. Δηλαδή, $[a, b] < [c, d]$, αρκεί $a < c$ ή ($a = c$ και $b < d$).

(ε) Μια διαδικασία `operator<` για έξοδο στην οθόνη.

7.2 Στην Άσκηση 7.1 δημιουργήσατε μια κλάση `Interval`. Χρησιμοποιήστε αυτή την κλάση για να εξετάσετε το ακόλουθο πρόβλημα. Έστω I_1, I_2, …, I_n μια συλλογή τυχαίων διαστημάτων. Ποια είναι η πιθανότητα ότι ένα από αυτά τα διαστήματα τέμνει όλα τα άλλα;

C++ για Μαθηματικούς

Μια εισαγωγή για σπουδαστές και καθηγητές

Στις περιπτώσεις προβλημάτων που απαιτούν εκτεταμένους υπολογισμούς, ένα πρόγραμμα C++ μπορεί να διατρέξει δισεκατομμύρια παραδειγμάτων ταχύτερα από τις περισσότερες άλλες επιλογές υπολογισμού. Η C++ δίνει τη δυνατότητα στους μαθηματικούς να δημιουργούν προγράμματα πολύ γρήγορα, ενώ είναι διαθέσιμη στα περισσότερα συστήματα υπολογιστών χωρίς επιπλέον κόστος. Το βιβλίο **C++ για Μαθηματικούς: Μια εισαγωγή για σπουδαστές και καθηγητές** δίνει έμφαση σε έννοιες της C++ που είναι πολύτιμες τόσο για τη θεωρητική όσο και για την εφαρμοσμένη μαθηματική έρευνα.

Αυτό το βιβλίο είναι το πρώτο που κυκλοφορεί για τον προγραμματισμό σε C++ και απευθύνεται ειδικά σε μαθηματικό ακροατήριο· παραλείπει τα «σκοτεινότερα» χαρακτηριστικά της γλώσσας προκειμένου να αναδείξει εκείνες τις πλευρές της που έχουν μεγαλύτερη χρηστική αξία για μαθηματικές εργασίες. Ο συγγραφέας εξηγεί πώς να χρησιμοποιείτε τη C++ για να διατυπώνετε εικασίες, πώς να δημιουργείτε εικόνες και διαγράμματα, πώς να επαληθεύετε αποδείξεις, πώς να κατασκευάζετε μαθηματικές δομές, και πώς να εξερευνάτε μυριάδες παραδειγμάτων. Επειδή ακριβώς δίνει έμφαση στον ουσιαστικό ρόλο της *πρακτικής εξάσκησης* κατά τη διαδικασία εκμάθησης, το βιβλίο είναι ιδανικό για προπτυχιακούς φοιτητές — αλλά και για προσωπική μελέτη. Σε κάθε κεφάλαιο περιλαμβάνονται πολλά προβλήματα και λύσεις, έτσι ώστε ο αναγνώστης να θεμελιώσει καλύτερα τις γνώσεις του, ενώ το συνοδευτικό CD-ROM περιέχει όλα τα αριθμημένα προγράμματα ώστε ο αναγνώστης να μπορεί να τα χρησιμοποιήσει αμέσως ή να τα προσαρμόσει στις ανάγκες του.

Χαρακτηριστικά του βιβλίου

- Παρουσιάζει τη γλώσσα προγραμματισμού C++ μέσω παραδειγμάτων που αναδεικνύουν τον τρόπο με τον οποίο χρησιμοποιείται η C++ στη μαθηματική έρευνα
- Περιγράφει πώς μπορεί να χρησιμοποιηθεί η C++ σε διαφορετικά συστήματα υπολογιστών, έτσι ώστε ο αναγνώστης να είναι σε θέση να αρχίσει τον προγραμματισμό αμέσως
- Παρέχει μια σαφή και περιεκτική εισαγωγή της αντικειμενοστρεφούς μεθόδου προγραμματισμού
- Εξετάζει διάφορα σχετικά θέματα — όπως την είσοδο/έξοδο, την οπτική αναπαράσταση, και την τεκμηρίωση — καθώς και επιλεγμένα ειδικά χαρακτηριστικά της γλώσσας C++
- Περιλαμβάνει έναν οδηγό γρήγορης αναφοράς στη γλώσσα C++

Παρουσιάζοντας σαφείς εξηγήσεις και παραδείγματα από τον κόσμο των μαθηματικών, και αναπτύσσοντας τις έννοιες από τη βάση τους, το βιβλίο **C++ για Μαθηματικούς** μπορεί να χρησιμοποιηθεί ανά πάσα στιγμή ως πηγή πληροφοριών για την εφαρμογή της C++ σε κάθε είδους προβλήματα — από τα πλέον βασικά έως τα πλέον πολύπλοκα.

Επισκεφθείτε μας στο Internet:
www.klidarithmos.gr

εκδόσεις
ΚΛΕΙΔΑΡΙΘΜΟΣ
Δουκάτου 4, Στοά Βιβλίου Λόρεντζος, 10440 ΑΘΗΝΑ, Τηλ. 210-5237505

ISBN 978-960-461-322-9