Εισαγωγή στη βιβλιοθήκη STL

Ένα πολύ σημαντικό τμήμα των standard βιβλιοθηκών της C++ είναι η βιβλιοθήκη STL (Standard Template Library). Αυτή βασίζεται στα templates: τον τρόπο της C++ να περιγράφει γενικό (generic) κώδικα που ενεργεί πάνω σε κάθε τύπο δεδομένων, αντί μόνο σε έναν (αντί π.χ. μόνο πάνω σε int).

Η συγγραφή κώδικα με templates είναι πολύπλοκη, το ίδιο και η μεταγλώττισή του. Ευτυχώς, **η χρήση** βιβλιοθηκών με templates είναι πολύ εύκολη, όπως θα φανεί στη συνέχεια!

Τα βασικά μέρη της βιβλιοθήκης STL είναι:

- **STL Containers:** δομές δεδομένων για την αποθήκευση και διαχείριση δεδομένων του χρήστη. Τα δεδομένα μπορούν να είναι οποιουδήποτε τύπου, native ή user-defined.
- **STL Iterators:** "δείχνουν" σε κάποιο στοιχείο αποθηκευμένο σε container και επιτρέπουν τον χειρισμό του.
- **STL Algorithms:** κοινές λειτουργίες όπως αναζήτηση, ταξινόμηση κ.ά. σε containers. Οι λειτουργίες αυτές είναι ανεξάρτητες από το είδος του container, γιατί χρησιμοποιούν iterators.

Note

Λεπτομέρειες θα δούμε στη συνέχεια, προς το παρόν σημειώνεται ότι: η βιβλιοθήκη STL έχει σχεδιαστεί και υλοποιηθεί από ειδικούς, έτσι ώστε να επιτυγχάνει τη βέλτιστη απόδοση. Πάντοτε πρέπει να τη χρησιμοποιούμε αντί να γράφουμε εκ νέου τον δικό μας κώδικα για λειτουργίες που παρέχει ήδη η STL.

STL Vectors: Οδηγίες Χρήσης

δύο είναι οι βασικές κατηγορίες STL containers:

- **Ακολουθιακά containers**, μια αλληλουχία αντικειμένων (όπως ένας πίνακας, μια λίστα, μια ουρά..).
- **Προσεταιριστικά containers**, μια διατεταγμένη αντιστοιχία ζευγαριών <Κλειδί-Τιμή> (όπως ένα λεξικό).

Τα vectors είναι τα πιο απλά και περισσότερο χρησιμοποιούμενα ακολουθιακά containers: Μονοδιάστατοι πίνακες για την αποθήκευση αντικειμένων σε σειρά με δυναμικό μέγεθος, διαχειριζόμενο από τη βιβλιοθήκη STL. Ο χρήστης δεν ασχολείται με το μέγεθος ένος vector, αυτό προσαρμόζετια αυτόματα για να χωρά τα αντικείμενα που εισάγει ο χρήστης.

Για να χρησιμοποιήσουμε τα vectors πρέπει να προσθέσουμε στα includes:

```
#include <vector>
```

Στη συνέχεια μπορούμε να ορίσουμε μια μεταβλητή vector όπως συνήθως (προσέξτε τον τρόπο που δηλώνουμε ότι θέλουμε ένα vector που να περιέχει int):

```
vector<int> vi;      // an empty vector
if (vi.empty()) cout << "vector is empty" << endl;
cout << "size=" << vi.size() << " and capacity=" << vi.capacity() << endl;
```

Παρατηρήσεις:

- Όλες οι μεταβλητές τύπου STL container αρχικοποιούνται από τον constructor τους. Στο προηγούμενο παράδειγμα, το vi είναι empty αλλά αρχικοποιημένο.
- Η συνάρτηση-μέλος empty() επιστρέφει true αν το vector είναι άδειο.
- Η συνάρτηση-μέλος size() επιστρέφει το πλήθος των στοιχείων στο vector τη στιγμή αυτή.
- Η συνάρτηση-μέλος capacity() επιστρέφει το πλήθος των στοιχείων που χωράνε στο vector τη στιγμή αυτή (το vector θα "αναμορφωθεί" αυτόματα για να χωρέσει πρόσθετα στοιχεία).

Μπορούμε επίσης να αρχικοποιήσουμε το vector με έναν αριθμό στοιχείων:

```
vector<double> vd1(10,0.23); // 10 items, 0.23 each
vector<double> vd2(10);      // 10 items, of "double()" value (that is, 0)
vector<double> vd3 = vd2;    // copy constructor
```

Στη δεύτερη περίπτωση, καλείται ο default constructor για την αρχικοποίηση κάθε στοιχείου του vector, αν το στοιχείο είναι τύπου native (άρα δεν έχει constructor) τότε η αρχικοποίηση είναι στην τιμή 0. Στην τρίτη περίπτωση δημιουργείται ένα νέο vector αντιγράφοντας τα στοιχεία ενός προϋπάρχοντος.

Πολλές φορές ξέρουμε ότι θα χρειαστούμε έναν πίνακα N στοιχείων (π.χ. N=1000) αλλά δεν θέλουμε αρχικοποίηση. Τότε είναι πιο αποδοτικό να ορίσουμε ένα empty vector και να υποδείξουμε ότι θα πρέπει να χωρά N στοιχεία. Αυτό εμποδίζει τις χρονοβόρες αυτόματες αλλαγές μεγέθους του vector:

```
vector<int> vi;      // an empty vector
vi.reserve(1000);
```

Τα vectors είναι πολύ αποδοτικά στην τυχαία προσπέλαση του i-οστού στοιχείου και την προσθήκη νέου στοιχείου στο τέλος του vector (αν θέλετε αποδοτικές εισαγωγές-διαγραφές πρέπει να χρησιμοποιήσετε άλλον τύπο container!). Η εισαγωγή στο τέλος γίνεται ως εξής:

```
vector<int> vi;

vi.push_back(3);
vi.push_back(13);
vi.push_back(23);
```

Η προσπέλαση γίνεται με τον τελεστή [] για ανάγνωση και εγγραφή:

```
cout << vi[0];
vi[1] = 7;
```

Note

ΠΡΟΣΟΧΗ!!! Ο τελεστής [] **δεν ελέγχει** αν προσπαθείτε να διαβάσετε ή να γράψετε πέρα από το τρέχον μέγιστο στοιχείο του vector! Αυτό φυσικά θα είναι μάλλον καταστροφικό για το προγραμμά σας.. Αν φοβάστε ότι κάτι τέτοιο μπορεί να συμβεί, χρησιμοποιήστε τη συνάρτηση-μέλος at(i), η οποία "ρίχνει" ένα exception αν το i είναι εκτός των ορίων του vector. Τόσο το [i] όσο και το at(i) επιστρέφουν μια αναφορά (reference) στο i-οστό στοιχείο του vector.

Με τη βοήθεια του [] μπορείτε να τυπώσετε τα περιεχόμενα του vector ως εξής (η μέθοδος αυτή δεν συνηθίζεται όμως! Για μια πιο "επαγγελματική" μέθοδο δείτε στα επόμενα):

```
// cycle trough vector via index
for (unsigned int i=0; i<vd1.size(); ++i) {
  cout << vd1[i] << " ";
}
cout << endl;
```

Η "επαγγελματική" μέθοδος χρησιμοποιεί iterators:

```
vector<double> vd2(10,0.23); // 10 items, 0.23 each

// cycle through vector via iterator
for (vector<double>::iterator vdp=vd2.begin(); vdp!=vd2.end(); ++vdp) {
  cout << *vdp << " ";
}
cout << endl;
```

Παρατηρήσεις:

- Ο iterator πρέπει να έχει τον αντίστοιχο τύπο με το vector: εδώ είναι `vector<double>::iterator`.
    - Μια const παραλλαγή (`vector<double>::const_iterator`) επιτρέπει την ανάγνωση αλλά όχι και την εγγραφή ενός στοιχείου.

- Το στοιχείο που "δείχνει" ο iterator προσπελαύνεται μέσω του τελεστή *
- Η συνάρτηση-μέλος begin() επιστρέφει έναν iterator στο πρώτο στοιχείο του vector.
- Η συνάρτηση-μέλος end() επιστρέφει έναν iterator **αμέσως μετά το τελευταίο** στοιχείο του vector.

  Note

  Ποτέ!! δεν προσπελαύνουμε το στοιχείο που δείχνει το end(), παρά μόνο έως το προηγούμενο από αυτό!

- Η αύξηση του iterator κατά 1 (++vdp) έχει ως συνέπεια ο iterator να δείχνει στο επόμενο στοιχείο του vector.