ΔΕΚΑΤΗ ΔΙΑΛΕΞΗ

Συνιστώσες της Standard Library

Η Standard Library έχει τρεις κύριες συνιστώσες:

- τους containers, δομές με κατάλληλα χαρακτηριστικά για την αποθήκευση και διαχείριση δεδομένων οποιουδήποτε τύπου, η κάθε μία με διαφορετικές ιδιότητες και δυνατότητες.
- τους iterators, ένα είδος δείκτη σε θέσεις μιας συλλογής στοιχείων (π.χ. ενός container, ενός string ή ενός αρχείου). Οι iterators έχουν την ίδια μορφή για όλες τις ακολουθίες στοιχείων με αποτέλεσμα να παρέχουν συγκεκριμένο, ενιαίο τρόπο για τη διαχείρισή τους.
- τους αλγόριθμους, συναρτήσεις που υλοποιούν με πολύ αποτελεσματικό τρόπο τμήματα κώδικα που χρειάζονται συχνά (π.χ. ταξινόμηση στοιχείων μιας ακολουθίας, αναζήτηση ή αντικατάσταση στοιχείου με συγκεκριμένη τιμή σε αυτή κλπ.). Είναι ανεξάρτητοι από τη δομή αποθήκευσης καθώς δρουν σε iterators.

Iterators: Κατηγορίες (1/5)

Iterator εισόδου

Ένας iterator εισόδου μπορεί να χρησιμοποιηθεί για να *διαβάσουμε* (μόνο) την τιμή στη θέση που δείχνει. Κάθε ανάγνωση τιμής πρέπει να ακολουθείται από μετατόπιση (και όχι νέα ανάγνωση τιμής).
Για τέτοιο iterator με όνομα it

- δεν επιτρέπεται η χρήση του `*it` στο αριστερό μέλος εντολής εκχώρησης.
- επιτρέπεται η δράση του τελεστή '`->`' για ανάγνωση μέλους στοιχείου.
- επιτρέπεται η μετακίνηση του it μόνο κατά μία θέση και μόνο προς τα εμπρός.
- επιτρέπεται η σύγκριση για ισότητα ή μη, με iterator που δείχνει σε μία θέση μετά το τέλος μιας ακολουθίας στοιχείων. Η δυνατότητα σύγκρισης μπορεί να μην ορίζεται για iterators σε άλλες θέσεις.

Iterator εισόδου μπορεί να συνδεθεί με ροή εισόδου (π.χ. αρχείο για ανάγνωση).

Iterators: Κατηγορίες (2/5)

Iterator εξόδου

Ένας iterator εξόδου μπορεί να χρησιμοποιηθεί για να *δώσουμε* (μόνο) τιμή στη θέση που δείχνει. Κάθε εκχώρηση τιμής πρέπει να εναλλάσσεται με μετατόπιση.
Για τέτοιο iterator με όνομα it

- δεν επιτρέπεται η χρήση του `*it` για ανάγνωση τιμής.
- επιτρέπεται η δράση του ++ για μετακίνηση του it μόνο κατά μία θέση και μόνο προς τα εμπρός.

Iterator εξόδου μπορεί να συνδεθεί με ροή εξόδου (π.χ. αρχείο για εγγραφή).

Iterators: Κατηγορίες (3/5)

Iterator μονής κατεύθυνσης

Ένας iterator μονής κατεύθυνσης (από την αρχή προς το τέλος της ακολουθίας τιμών)

- δίνει πρόσβαση στην τιμή της θέσης που δείχνει με τη δράση του τελεστή '$*$' ή σε μέλος του στοιχείου με τον τελεστή '$->$'.
- μπορεί να μετακινηθεί κατά μία θέση, μόνο προς τα εμπρός.
- μπορεί να συγκριθεί με άλλο iterator αυτής της κατηγορίας μόνο για ισότητα ή μη.
- έχει όλες τις ιδιότητες του iterator εισόδου και μπορεί να συμπεριφερθεί ως τέτοιος.
- μπορεί να διαβάσει την ίδια τιμή πολλές φορές.

Οι iterators για το `std::forward_list<>` και τους unordered associative containers είναι αυτού του είδους.

Iterators: Κατηγορίες (4/5)

Iterator διπλής κατεύθυνσης

Ένας iterator διπλής κατεύθυνσης έχει όλες τις ιδιότητες των iterators μονής κατεύθυνσης και μπορεί να συμπεριφερθεί ως τέτοιος. Επιπλέον, επιτρέπεται να μετακινηθεί κατά μία θέση προς τα πίσω, με τον τελεστή '`--`'.

Οι iterators για το `std::list<>` και τους associative containers (`std::set<>`, `std::multiset<>`, `std::map<>`, `std::multimap<>`) είναι αυτού του είδους.

Iterators: Κατηγορίες (5/5)

Iterator τυχαίας προσπέλασης

Ένας iterator τυχαίας προσπέλασης έχει όλες τις ιδιότητες του iterator διπλής κατεύθυνσης και μπορεί να συμπεριφερθεί ως τέτοιος.
Επιπλέον, μπορούμε

- να του προσθέσουμε ή αφαιρέσουμε ένα ακέραιο αριθμό και να παραγάγουμε έτσι νέο iterator, μετατοπισμένο σε επόμενη ή προηγούμενη θέση.
- να τον μετακινήσουμε με τους σύνθετους τελεστές '+=' και '−='.
- να αφαιρέσουμε ένα iterator τυχαίας προσπέλασης από άλλον ώστε να υπολογίσουμε την απόστασή τους.
- να τον συγκρίνουμε με άλλο όμοιο iterator με όλους τους τελεστές σύγκρισης.

Οι iterators των `std::array<>`, `std::vector<>`, `std::deque<>` και `std::string` είναι τέτοιοι.

Βοηθητικές συναρτήσεις για iterators (1/3)

advance()

Η συνάρτηση

```
template<typename InputIterator, typename Distance>
void advance(InputIterator & it, Distance n);
```

δέχεται έναν iterator εισόδου, it, και μια ποσότητα ακέραιου τύπου, n. Αν ο it είναι στην πραγματικότητα

τυχαίας προσπέλασης, η κλήση της ισοδυναμεί με `it += n;`.

διπλής κατεύθυνσης, η κλήση της ισοδυναμεί με n διαδοχικές κλήσεις της εντολής `++it` (αν $n > 0$) ή `--it` (αν $n < 0$).

μονής κατεύθυνσης ή εισόδου, η κλήση της έχει νόημα μόνο αν το n δεν είναι αρνητικό και ισοδυναμεί με n διαδοχικές κλήσεις του `++it`.

Παρέχεται από το header `<iterator>`.

Βοηθητικές συναρτήσεις για iterators (2/3)

distance()

Η συνάρτηση

```
template<typename InputIterator>
Dist distance(InputIterator it1, InputIterator it2);
```

δέχεται δύο iterators it1 και it2, ίδιου τύπου, που δείχνουν στον ίδιο container.

- Αν οι iterators it1 και it2 είναι τυχαίας προσπέλασης, η συνάρτηση επιστρέφει το `it2-it1`.
  Σε άλλη περίπτωση αυξάνει το τοπικό αντίγραφο του ορίσματος it1 έως ότου γίνει ίσο με it2 και επιστρέφει το πλήθος των αυξήσεων.
  Προφανώς, πρέπει ο it1 να μη δείχνει μετά τον it2.
- Ο τύπος Dist είναι ακέραιος.

Η συνάρτηση παρέχεται από το header `<iterator>`.

Βοηθητικές συναρτήσεις για iterators (3/3)

next() και prev()

Οι συναρτήσεις

```
template<typename ForwardIterator>
ForwardIterator next(ForwardIterator it, Dist n = 1);

template<typename BidirectionalIterator>
BidirectionalIterator prev(BidirectionalIterator it, Dist n = 1);
```

ουσιαστικά καλούν την `std::advance()` και επιστρέφουν iterator, n θέσεις μετά/πριν τον it (χωρίς να τροποποιούν τον it). Αν το n είναι αρνητικό, ο iterator που επιστρέφεται είναι πριν/μετά τον it. Στη `std::next()` και για αρνητικό n, ο it πρέπει να είναι διπλής κατεύθυνσης.
Αν δεν προσδιοριστεί τιμή για το n, αυτό παίρνει την τιμή 1.
Ο τύπος Dist είναι ακέραιος.
Παρέχονται από το header `<iterator>`.

Ανάστροφοι iterators (1/2)

Κάθε container που έχει iterators διπλής κατεύθυνσης ή πιο γενικούς παρέχει και *ανάστροφους iterators*.
Οι ανάστροφοι iterators

- διατρέχουν ένα container *ανάστροφα*, από το τελευταίο στοιχείο προς το πρώτο, με την ίδια ευχρηστία που έχουν οι iterators «ορθής φοράς».
- διακρίνονται ανάλογα με το αν επιτρέπουν ή όχι την τροποποίηση των στοιχείων στα οποία δείχνουν. Οι τύποι τέτοιων iterators, π.χ. για ένα `std::vector<double>`, είναι

  `std::vector<double>::reverse_iterator`

  και

  `std::vector<double>::const_reverse_iterator`

  αντίστοιχα.

Ανάστροφοι iterators (2/2)

Συναρτήσεις παραγωγής ανάστροφων iterators

Κάθε container που έχει iterators διπλής κατεύθυνσης ή πιο γενικούς παρέχει σχετικές συναρτήσεις–μέλη:

- Οι `rbegin()` και `rend()` επιστρέφουν `reverse_iterator` στο *τελευταίο* στοιχείο και *σε μία θέση πριν το πρώτο στοιχείο* αντίστοιχα.
- Οι `crbegin()` και `crend()` επιστρέφουν `const_reverse_iterator` στο *τελευταίο* στοιχείο και *σε μία θέση πριν το πρώτο στοιχείο* αντίστοιχα.

Προσέξτε ότι οι τελεστές '++', '−−', όταν δρουν σε ανάστροφο iterator (σε σταθερό στοιχείο ή όχι), τον μετακινούν σε μία θέση *προς την αρχή* ή *προς το τέλος* του container αντίστοιχα.

Παράδειγμα

```
for (auto it = v.crbegin(); it != v.crend(); ++it) {
   std::cout << *it << '\n';
}
```

Ο κώδικας τυπώνει τα στοιχεία του container v με ανάστροφη σειρά.

Iterator για ροή

Ροή εισόδου

Μια ροή εισόδου συνδεδεμένη π.χ. με αρχείο, με τιμές *ίδιου τύπου*, μπορεί να προσαρμοστεί σε iterator. Ο iterator στην αρχή της είναι

```
std::istream_iterator<τύπος> όνομα{ροή};
```

ενώ στο τέλος της είναι

```
std::istream_iterator<τύπος> όνομα{};
```

Ροή εξόδου

Μια ροή εισόδου προσαρμόζεται σε iterator ως εξής

```
std::ostream_iterator<τύπος> όνομα{ροή, σειρά};
```

Το δεύτερο, προαιρετικό όρισμα, η "σειρά", ένα σύνολο χαρακτήρων εντός διπλών εισαγωγικών, τυπώνεται στη ροή μετά από κάθε εκτύπωση τιμής.

Οι προσαρμογείς `std::istream_iterator<>` και `std::ostream_iterator<>` παρέχονται από το header `<iterator>`.

Παράδειγμα χρήσης iterators (1/4)

Έστω ότι θέλουμε να γράψουμε μια συνάρτηση που θα αντιγράφει το πρώτο, τρίτο, πέμπτο κλπ. στοιχείο σε ένα διάστημα κάποιου container, σε διαδοχικές θέσεις κάποιου άλλου. Ας την ονομάσουμε copyodd. Τα διαστήματα θα προσδιορίζονται από iterators.
Αν επιθυμούμε να αντιγράψουμε κάθε δεύτερο στοιχείο του container a στον container b, μετά την όγδοη θέση του, θα πρέπει να μπορούμε να γράψουμε

```
copyodd(a.cbegin(), a.cend(), std::next(b.begin(), 8));
```

Η δήλωση της συνάρτησης είναι

```
template<typename Iterator1, typename Iterator2>
void copyodd(Iterator1 beg1, Iterator1 end1, Iterator2 beg2);
```

Παράδειγμα χρήσης iterators (2/4)

Μια πρώτη απόπειρα να γράψουμε τον ορισμό της συνάρτησης είναι

```
template<typename Iterator1, typename Iterator2>
void copyodd(Iterator1 beg1, Iterator1 end1, Iterator2 beg2)
{
  while (beg1 < end1) {
    *beg2 = *beg1;
    if (beg1 == end1 - 1) {
      break;
    }
    beg1+=2;
    ++beg2;
  }
}
```

Ο τύπος των beg1, end1 είναι αναγκαστικά iterator τυχαίας προσπέλασης. Ο beg2 απαιτείται να είναι τουλάχιστον iterator εξόδου.

Παράδειγμα χρήσης iterators (3/4)

Ένας άλλος τρόπος να γράψουμε τη συνάρτηση είναι ο εξής

```
#include <iterator>

template<typename Iterator1, typename Iterator2>
void copyodd(Iterator1 beg1, Iterator1 end1, Iterator2 beg2)
{
  while (beg1 != end1) {
    *beg2 = *beg1;
    if (std::next(beg1) == end1) {
      break;
    }
    std::advance(beg1, 2);
    ++beg2;
  }
}
```

Η συνάρτηση στην τωρινή της εκδοχή απαιτεί ο τύπος Iterator1 να είναι iterator μονής κατεύθυνσης τουλάχιστον, και ο τύπος Iterator2 να είναι iterator εξόδου ή επόμενος στην ιεραρχία.

Παράδειγμα χρήσης iterators (4/4)

Εναλλακτικά, μπορούμε να γράψουμε τη συνάρτηση ως εξής

```
template<typename Iterator1, typename Iterator2>
void copyodd(Iterator1 beg1, Iterator1 end1, Iterator2 beg2)
{
  while (beg1 != end1) {
    *beg2 = *beg1;
    ++beg1;
    if (beg1 == end1) {
      break;
    }
    *beg1;
    ++beg1;
    ++beg2;
  }
}
```

Οι beg1, end1 αρκεί πλέον να είναι iterator εισόδου.
Προσέξτε ότι χρειάστηκε να κάνουμε δύο προωθήσεις του beg1 και ενδιάμεσα μία ανάγνωση τιμής στο στοιχείο που δείχνει αυτός.

Ορισμός container

Έστω `cntr<T>` ένας οποιοσδήποτε τύπος container. Μπορούμε να ορίσουμε αντικείμενο αυτού του τύπου

- Κενό (χωρίς στοιχεία):

```
cntr<T> c1;
```

- Ως αντίγραφο άλλου:

```
cntr<T> c1;
// fill c1 ....
cntr<T> c2{c1};
```

- Από αρχικές τιμές:

```
cntr<T> c1{v1, v2, v3, ... };
```

- Για κάθε container εκτός από `std::array<>`, από τα στοιχεία που ορίζονται από ζεύγος iterators, [beg,end):

```
cntr<T> c1{beg,end};
```

Ορισμός container: Παραδείγματα

Παράδειγμα Α'

```
#include <vector>
#include <set>

std::vector<int> v1{0,2,-2,4,-4,6,-6};
std::vector<int> v2{v1};
std::set<int> s1{v1.cbegin(), v1.cend()};
```

Παράδειγμα Β'

Η δημιουργία ενός `std::vector<int>` με στοιχεία τους ακέραιους που παρατίθενται στο αρχείο "data" μπορεί να γίνει ως εξής

```
std::ifstream in{"data"};
std::istream_iterator<int> b{in}, e{};
std::vector<int> v{b,e};
```

Τροποποίηση στοιχείων container (1/2)

Σε οποιοδήποτε container, τροποποίηση στοιχείων γίνεται ως εξής:

- Με εκχώρηση άλλου container, ίδιου τύπου:

  ```
  c1 = c2;
  ```

  Τα αρχικά στοιχεία του c1 καταστρέφονται.
- Με εκχώρηση λίστας στοιχείων:

  ```
  c = {v1,v2,v3, ...};
  ```

  Τα αρχικά στοιχεία του c καταστρέφονται.
- Με κατάλληλους αλγόριθμους.

Τροποποίηση στοιχείων container (2/2)

Προσπέλαση στοιχείων, και άρα δυνατότητα τροποποίησής τους, έχουμε

- σε sequence container, με το μηχανισμό των iterators (`*it = ...`).
- σε `array<>`, `vector<>`, `deque<>`, με τον τελεστή '`[]`'.
- σε sequence container, με τις συναρτήσεις–μέλη `front()` και `back()`, που επιστρέφουν αναφορές στο πρώτο και στο τελευταίο στοιχείο.

Εισαγωγή ή διαγραφή στοιχείων container

Εισαγωγή νέων στοιχείων κάνουμε με τη συνάρτηση–μέλος `insert()` κάθε container (ή `insert_after()` για `forward_list<>`):

```
c.insert(pos, elem);
```

Διαγραφή στοιχείων γίνεται:

- Με τη συνάρτηση–μέλος `clear()`. Αυτή διαγράφει όλα τα στοιχεία αφήνοντας κενό τον container για τον οποίο καλείται.

  ```
  c.clear();
  ```

- Με τη χρήση της συνάρτησης–μέλους `erase()` με όρισμα ένα iterator ή ζεύγος iterators:

  ```
  c.erase(pos);
  c.erase(beg,end);
  ```

- Με κατάλληλους αλγόριθμους.

Κοινοί τύποι–μέλη των containers

Εκτός από τους iterators (`iterator`, `const_iterator` και για ορισμένους containers `reverse_iterator`, `const_reverse_iterator`) παρέχονται μεταξύ άλλων οι ακόλουθοι τύποι:

- `value_type`, που είναι ο τύπος των στοιχείων που αποθηκεύει,
- `reference`, που είναι ο τύπος της αναφοράς στα στοιχεία,
- `const_reference`, που είναι ο τύπος της αναφοράς σε σταθερά στοιχεία,
- `difference_type`, που είναι εμπρόσημος τύπος για διαφορές θέσεων των στοιχείων,
- `size_type`, που είναι απρόσημος ακέραιος τύπος για την αρίθμηση (και το πλήθος) των θέσεων αποθήκευσης.

Οι τύποι που ορίζονται σε containers μπορούν να χρησιμοποιηθούν για τη δήλωση σχετικών ποσοτήτων. Στη δήλωση γράφουμε τον τύπο του container, κατόπιν τον τελεστή εμβέλειας '::', και μετά το όνομα του τύπου.

Κοινές συναρτήσεις–μέλη των containers

size() Επιστρέφει το πλήθος των στοιχείων κατά τη στιγμή της κλήσης. Δεν ορίζεται για `std::forward_list<>`.

max_size() Επιστρέφει το μέγιστο δυνατό πλήθος στοιχείων.

empty() Επιστρέφει λογική τιμή, **`false`**/**`true`**, αν ο container είναι κενός ή όχι.

Παράδειγμα

Ο κατάλληλος τύπος για αρίθμηση των στοιχείων ενός `std::vector<int>` είναι ο `std::vector<int>::size_type`. Η εκτύπωση όλων των στοιχείων ενός `std::vector<int>` με όνομα v μπορεί να γίνει ως εξής

```
for (std::vector<int>::size_type i{0}; i < v.size(); ++i) {
   std::cout << v[i] << '\n';
}
```

Δημιουργία `std::vector<>`

Εκτός από τους κοινούς τρόπους δημιουργίας (κενό, αντίγραφο, από τιμές, από ζεύγος iterators), υπάρχουν ακόμη:

- Η εντολή

  ```
  std::vector<T> v(N);
  ```

  που ορίζει το v ως ένα `std::vector<>` με N θέσεις για αντικείμενα τύπου T. Αρχική τιμή το 0 (για αριθμητικούς τύπους).

- Η εντολή

  ```
  std::vector<T> v(N, elem);
  ```

  που ορίζει το v ως ένα `std::vector<>` με N αντίγραφα του αντικειμένου elem.

Εισαγωγή/τροποποίηση στοιχείων σε `std::vector<>` (1/2)

Εκτός από τους μηχανισμούς που είδαμε, μπορούμε να κάνουμε:

- Ταυτόχρονη εισαγωγή ενός πλήθους νέων στοιχείων:

  ```
  v.insert(pos, N, elem);
  v.insert(pos, beg, end);
  v.insert(pos, {a1, a2, a3, ...});
  ```

  Η πρώτη μορφή εισάγει N αντίγραφα του elem, *πριν* τη θέση που δείχνει ο `const_iterator` pos. Στη δεύτερη μορφή, εισάγονται στο v πριν τη θέση pos, αντίγραφα των στοιχείων στο διάστημα των iterators [beg,end). Στην τρίτη μορφή, εισάγονται στο v πριν τη θέση pos, αντίγραφα των στοιχείων της λίστας στο δεύτερο όρισμα.

- Εισαγωγή ενός στοιχείου στο τέλος:

  ```
  v.push_back(elem);
  ```

  Πρακτικά ισοδυναμεί με την `v.insert(v.cend(),elem);`.

Εισαγωγή/τροποποίηση στοιχείων σε `std::vector<>` (2/2)

- Αντικατάσταση όλων των στοιχείων:

```
v.assign(N,elem);
v.assign(beg, end);
v.assign({a1,a2,a3,...});
```

  Η πρώτη μορφή εισάγει N αντίγραφα του elem στο `vector<>` v.
  Στη δεύτερη μορφή, αντικαθιστά τα στοιχεία του v με αντίγραφα των στοιχείων στο διάστημα [beg,end).
  Στην τρίτη μορφή, η κλήση εισάγει τις τιμές της λίστας.

- Αλλαγή του πλήθους σε N:

```
v.resize(N);
```

  Διαγράφονται από το τέλος ή προστίθενται εκεί στοιχεία. Τα στοιχεία που προστίθενται έχουν την προκαθορισμένη τιμή για τον τύπο τους. Με δεύτερο όρισμα, δηλαδή με την εντολή

```
v.resize(N,elem);
```

  τα τυχόν νέα στοιχεία είναι αντίγραφα του elem.

- Διαγραφή του τελευταίου στοιχείου:

```
v.pop_back();
```