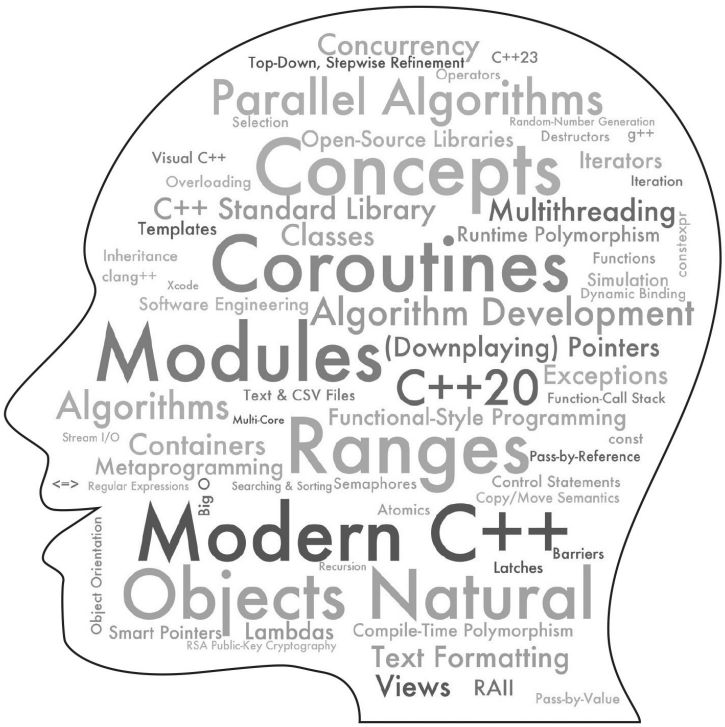
Δομές Δεδομένων: Κοντέινερ και Επαναλήπτες της Πρότυπης Βιβλιοθήκης

13



Στόχοι

Σε αυτό το κεφάλαιο:

- Θα μάθετε περισσότερα σχετικά για τα επαναχρησιμοποιήσιμα κοντέινερ, επαναλήπτες και αλγόριθμους της πρότυπης βιβλιοθήκης της C++.
- Θα κατανοήσετε πώς σχετίζονται τα κοντέινερ με τα εύρη στην C++20.
- Θα χρησιμοποιήσετε επαναλήπτες ροής εισόδου/εξόδου για να διαβάσετε τιμές από την τυπική ροή εισόδου και να γράψετε τιμές στην τυπική ροή εξόδου.
- Θα χρησιμοποιήσετε επαναλήπτες για πρόσβαση σε στοιχεία ενός κοντέινερ.
- Θα χρησιμοποιήσετε τα κοντέινερ ακολουθίας `vector`, `list` και `deque`.
- Θα χρησιμοποιήσετε `ostream_iterator` με τους αλγόριθμους `std::copy` και `std::ranges::copy` για να εξάγετε στοιχεία από ένα κοντέινερ σε μια μόνο πρόταση.
- Θα χρησιμοποιήσετε τα διατεταγμένα κοντέινερ συσχέτισης `set` `multiset`, `map` και `multimap`.
- Θα κατανοήσετε τις διαφορές μεταξύ των διατεταγμένων και μη διατεταγμένων κοντέινερ συσχέτισης.
- Θα χρησιμοποιήσετε τους προσαρμογείς κοντέινερ `stack`, `queue` και `priority_queue`.
- Θα χρησιμοποιήσετε το «περίπου σαν κοντέινερ» `bitset` για να χειριστείτε μια συλλογή από σημαίες-bit.

Κύρια Σημεία

13.1 Εισαγωγή

Η πρότυπη βιβλιοθήκη ορίζει ισχυρά, βασισμένα σε πρότυπα, επαναχρησιμοποιήσιμα συστατικά που χειρίζονται πολλές κοινές δομές δεδομένων και αλγόριθμους που χρησιμοποιούνται για την επεξεργασία αυτών των δομών δεδομένων. Αρχίσαμε να παρουσιάζουμε τα πρότυπα στα Κεφάλαια 5–6 και τα χρησιμοποιούμε εκτενώς εδώ και στα Κεφάλαια 14 και 15. Ιστορικά, οι λειτουργίες που παρουσιάζονται σε αυτό το κεφάλαιο αναφέρονταν ως **Standard Template Library** ή **STL**[1]. Στο πρότυπο έγγραφο της C++, αναφέρονται απλώς ως μέρος της πρότυπης βιβλιοθήκης της C++.

Κοντέινερ, Επαναλήπτες και Αλγόριθμοι

Αυτό το κεφάλαιο παρουσιάζει τρία βασικά συστατικά της πρότυπης βιβλιοθήκης—τα **κοντέινερ** (πρότυπα δομών δεδομένων), τους επαναλήπτες και τους αλγόριθμους. Θα παρουσιάσουμε τα **κοντέινερ**, **προσαρμογείς κοντέινερ** και τα **σχεδόν κοντέινερ**.

Κοινές Συναρτήσεις-Μέλη Μεταξύ Κοντέινερ

Κάθε κοντέινερ έχει συσχετισμένες συναρτήσεις-μέλη – ένα υποσύνολο αυτών ορίζεται σε όλα τα κοντέινερ. Παρουσιάζουμε το μεγαλύτερο μέρος αυτής της κοινής λειτουργικότητας στα παραδείγματά μας με **`array`** (που παρουσιάσθηκαν στο Κεφάλαιο 6), **`vector`** (που παρουσιάσθηκαν επίσης στο Κεφάλαιο 6 και καλύπτονται σε μεγαλύτερο βάθος εδώ), **`list`** (Ενότητα 13.9) και **`deque`** (προφέρεται «ντεκ»; Ενότητα 13.10).

Επαναλήπτες

Οι **επαναλήπτες** (iterator), οι οποίοι έχουν ιδιότητες παρόμοιες με εκείνες των **δεικτών**, χρησιμοποιούνται για το χειρισμό στοιχείων ενός κοντέινερ. Τους **ενσωματωμένους πίνακες** μπορούν επίσης να τους χειριστούν οι αλγόριθμοι της πρότυπης βιβλιοθήκης, χρησιμοποιώντας δείκτες ως επαναλήπτες. Θα δούμε ότι ο χειρισμός κοντέινερ μέσω επαναληπτών παρέχει τεράστια εκφραστική δύναμη όταν συνδυάζεται με αλγόριθμους της πρότυπης βιβλιοθήκης – μερικές φορές περιορίζοντας πολλές γραμμές κώδικα σε μια μόνο πρόταση.

1. Το STL αναπτύχθηκε από τους Alexander Stepanov και Meng Lee στη Hewlett-Packard και βασίζεται στην αρχική τους έρευνα προγραμματισμού, με σημαντικές συνεισφορές από τον David Musser. `https://en.wikipedia.org/wiki/History_of_the_Standard_Template_Library`, Πρόσβαση στις 18 Απριλίου, 2023.

Αλγόριθμοι

Οι **αλγόριθμοι** της πρότυπης βιβλιοθήκης (τους οποίους θα καλύψουμε στο Κεφάλαιο 14) είναι πρότυπα συναρτήσεων που εκτελούν συνήθεις χειρισμούς δεδομένων, όπως **αναζήτηση**, **ταξινόμηση**, **αντιγραφή**, **μετασχηματισμό** και **σύγκριση στοιχείων** ή **ολόκληρων κοντέινερ**. Η πρότυπη βιβλιοθήκη παρέχει πάνω από 100 αλγορίθμους:

- 90 στον `std` χώρο ονομάτων της κεφαλίδας **`<algorithm>`**—82 είναι επίσης υπερφορτωμένοι στον **χώρο ονομάτων `std::ranges`** για χρήση με ένα εύρος της C++20,
- 11 στον `std` χώρο ονομάτων της κεφαλίδας **`<numeric>`**,
- 14 στον `std` χώρο ονομάτων της κεφαλίδας **`<memory>`**—και οι 14 είναι επίσης υπερφορτωμένοι στον χώρο ονομάτων `std::ranges` για χρήση με ένα εύρος της C++20 και
- 2 στην κεφαλίδα **`<cstdlib>`**.

Για την πλήρη λίστα των αλγορίθμων με συνδέσμους για τις περιγραφές τους, επισκεφθείτε την διεύθυνση:

```
https://en.cppreference.com/w/cpp/algorithm
```

Οι περισσότεροι αλγόριθμοι χρησιμοποιούν επαναλήπτες για πρόσβαση σε στοιχεία κοντέινερ. Κάθε αλγόριθμος έχει ελάχιστες απαιτήσεις για τα είδη επαναληπτών που μπορούν να χρησιμοποιηθούν μαζί του. Θα δούμε ότι τα κοντέινερ υποστηρίζουν συγκεκριμένα είδη επαναληπτών, μερικούς πιο ισχυρούς από άλλους. Οι επαναλήπτες που υποστηρίζει ένα κοντέινερ προσδιορίζουν εάν το κοντέινερ μπορεί να χρησιμοποιηθεί με έναν συγκεκριμένο αλγόριθμο. Οι επαναλήπτες ενσωματώνουν τους μηχανισμούς που χρησιμοποιούνται για τη διάσχιση των κοντέινερ και την πρόσβαση σε στοιχεία τους, επιτρέποντας την εφαρμογή πολλών αλγορίθμων σε κοντέινερ με σημαντικά διαφορετικούς χειρισμούς. Αυτό σας επιτρέπει επίσης να δημιουργήσετε νέους αλγόριθμους που μπορούν να επεξεργαστούν τα στοιχεία πολλών τύπων κοντέινερ.

Εύρη της C++20

Το Κεφάλαιο 6 παρουσίασε τα **εύρη** (range) και τις **προβολές** (view) της C++20. Είδατε ότι ένα **εύρος** είναι μια συλλογή στοιχείων που μπορείτε να διασχίσετε. Έτσι, ένα **`array`** και ένα **`vector`** είναι ένα εύρος. Χρησιμοποιήσατε επίσης **προβολές** για να καθορίσετε **διοχετεύσεις** λειτουργιών που χειρίζονται ένα εύρος από στοιχεία. Οποιοδήποτε κοντέινερ με επαναλήπτες που αντιπροσωπεύουν την αρχή και το τέλος του μπορεί να αντιμετωπιστεί ως ένα εύρος της C++20. Σε αυτό το κεφάλαιο, θα χρησιμοποιήσουμε τον νέο της πρότυπης βιβλιοθήκης της C++20 `std::ranges::copy` και τον παλαιότερο αλγόριθμο `std::copy` της πρότυπης βιβλιοθήκης της C++ για να δείξουμε πώς μπορεί ένα εύρος να απλοποιήσει τον κώδικά σας. Στο Κεφάλαιο 14, θα χρησιμοποιήσουμε πολλούς περισσότερους αλγόριθμους της C++20 από τον χώρο ονομάτων `std::ranges` για να δείξουμε πρόσθετες λειτουργίες των **range** και **view**.

Προσαρμοσμένες Δομές Δεδομένων με Πρότυπα

Ορισμένες δημοφιλείς δομές δεδομένων περιλαμβάνουν συνδεδεμένες λίστες, ουρές, στοίβες και δυαδικά δέντρα:

- Οι **συνδεδεμένες λίστες** είναι συλλογές στοιχείων δεδομένων λογικά «παραταγμένων σε μια σειρά». Οι εισαγωγές και οι διαγραφές γίνονται οπουδήποτε σε μια συνδεδεμένη λίστα.
- Οι **στοίβες** είναι σημαντικές για τους μεταγλωττιστές και τα λειτουργικά συστήματα. Οι εισαγωγές και οι διαγραφές γίνονται **μόνο** στο ένα άκρο μιας στοίβας—στην **κορυφή** της.
- Οι **ουρές** αντιπροσωπεύουν τις ουρές αναμονής. Οι εισαγωγές γίνονται στο πίσω μέρος (που επίσης ονομάζεται **ουρά**) και οι διαγραφές γίνονται από μπροστά (ονομάζεται επίσης **κεφαλή**).
- Τα **δυαδικά δέντρα** είναι μη γραμμικές, ιεραρχικές δομές δεδομένων που διευκολύνουν την **αναζήτηση** και **ταξινόμηση** δεδομένων και την **απαλοιφή διπλότυπων τιμών**.

Κάθε μια από αυτές τις δομές δεδομένων έχει πολλές άλλες ενδιαφέρουσες εφαρμογές. Μπορούμε προσεκτικά να συνδέσουμε αντικείμενα με δείκτες, αλλά ο κώδικας που βασίζεται σε δείκτες είναι περίπλοκος και μπορεί να είναι επιρρεπής σε σφάλματα. Οι παραμικρές παραλείψεις ή παρακάμψεις μπορούν να οδηγήσουν σε σοβαρές **παραβιάσεις πρόσβασης της μνήμης** και **διαρροές μνήμης** χωρίς προειδοποίηση από τον μεταγλωττιστή.

Αποφύγετε την επανεφεύρεση του τροχού. Όταν είναι δυνατόν, χρησιμοποιήστε προϋπάρχοντα κοντέινερ, επαναλήπτες και αλγόριθμους της πρότυπης βιβλιοθήκης της C++[2]. Οι έτοιμες κλάσεις κοντέινερ παρέχουν τις δομές δεδομένων που χρειάζεστε για τις περισσότερες εφαρμογές. Η χρήση των ελεγμένων κοντέινερ, επαναληπτών και αλγορίθμων της πρότυπης βιβλιοθήκης σάς βοηθά να μειώσετε το χρόνο των δοκιμών και του εντοπισμού σφαλμάτων. Συλλήφθηκαν και σχεδιάστηκαν για απόδοση και ευελιξία.

Αυτό το κεφάλαιο αρχίζει με δύο ειδικά τμήματα της επιστήμης των υπολογιστών σχετικά με

- την **σύνταξη Big O** που εκφράζει το πόσο σκληρά πρέπει να εργαστούν οι αλγόριθμοι με βάση τον αριθμό των στοιχείων που επεξεργάζονται, και
- τους πίνακες κατακερματισμού (hash) για αποθήκευση και ανάκτηση δεδομένων σε υψηλή ταχύτητα σε εφαρμογές με κοινά χαρακτηριστικά επαγγελματικών εφαρμογών.

13.2 Μια Σύντομη Εισαγωγή στο Big *O*

Σε αυτό το κεφάλαιο, θα συζητήσουμε τι σημαίνει η σύνταξη Big *O* και θα εξηγήσουμε τη σημασία πολλών δημοφιλών συμβολισμών Big *O*, συμπεριλαμβανομένων των **O(1)**, **O(n)**, **O(log n)** και **O(n²)**. Στο Κεφάλαιο 21, θα εξηγήσουμε το Big *O* του **O(n log n)**. Στους σημερινούς επιτραπέζιους υπολογιστές, οι οποίοι μπορεί να εκτελούν μερικά δισεκατομμύρια πράξεις ανά δευτερόλεπτο, η αποτελεσματικότητα ενός αλγορίθμου, όπως εκφράζεται από το Big *O* του, μεταφράζεται στο αν ένα πρόγραμμα θα εκτελεστεί σχεδόν ακαριαία ή θα χρειαστεί δευτερόλεπτα, λεπτά, ώρες, ημέρες, μήνες ή ακόμα και χρόνια για να ολοκληρωθεί. Προφανώς, θα προτιμούσατε αλγόριθμους που ολοκληρώνονται γρήγορα, παρόλο που είναι μεγάλος ο αριθμός των στοιχείων *n* που μπορεί να πρέπει να επεξεργασθούν. Αυτό ισχύει ιδιαίτερα στον σημερινό κόσμο των εφαρμογών Big Data. Θα αναφέρουμε εν συντομία το Big *O* εδώ, θα το χρησιμοποιήσουμε εκτενώς στο Κεφάλαιο 14 για να χαρακτηρίσουμε την αποτελεσματικότητα των αλγορίθμων της πρότυπης βιβλιοθήκης και θα το χρησιμοποιήσουμε στο Κεφάλαιο 21 για να χαρακτηρίσουμε την αποτελεσματικότητα διαφόρων δημοφιλών αλγορίθμων αναζήτησης και ταξινόμησης που θα χειριστούμε.

Αλγόριθμοι O(1)

Ας υποθέσουμε ότι ένας αλγόριθμος ελέγχει εάν το πρώτο στοιχείο ενός πίνακα είναι ίσο με το δεύτερο. Αν ο πίνακας έχει 10 στοιχεία, αυτός ο αλγόριθμος απαιτεί μια σύγκριση. Εάν ο πίνακας έχει 1.000 στοιχεία, εξακολουθεί να απαιτεί μια σύγκριση. Στην πραγματικότητα, ο αλγόριθμος είναι ανεξάρτητος από τον αριθμό, *n*, των στοιχείων του πίνακα. Αυτός ο αλγόριθμος λέγεται ότι έχει **σταθερό χρόνο εκτέλεσης**, που αντιπροσωπεύεται σύμφωνα με τη σύνταξη Big *O* ως **O(1)** και προφέρεται ως «τάξης ένα». Ένας αλγόριθμος που είναι $O(1)$ δεν σημαίνει απαραίτητα ότι απαιτεί μόνο μια σύγκριση. Το **O(1)** απλώς σημαίνει ότι ο αριθμός των συγκρίσεων είναι σταθερός—δεν αυξάνεται καθώς αυξάνεται το μέγεθος του πίνακα. Ένας αλγόριθμος που ελέγχει αν το πρώτο στοιχείο ενός πίνακα είναι ίσο με οποιοδήποτε από τα επόμενα τρία στοιχεία εξακολουθεί να είναι $O(1)$, παρόλο που απαιτεί τρεις συγκρίσεις.

Αλγόριθμοι O(n)

Ένας αλγόριθμος που ελέγχει εάν το πρώτο στοιχείο του πίνακα είναι ίσο με *οποιοδήποτε* από τα στοιχεία των άλλων πινάκων θα απαιτήσει το πολύ $n - 1$ συγκρίσεις, όπου *n* είναι ο αριθμός των στοιχείων του πίνακα. Εάν ο πίνακας έχει 10 στοιχεία, αυτός ο αλγόριθμος απαιτεί έως και

2. C++ Core Guidelines, "SL.1: Use Libraries Wherever Possible". Πρόσβαση στις 18 Απριλίου, 2023. `https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rsl-lib`.

εννέα συγκρίσεις. Εάν ο πίνακας έχει 1.000 στοιχεία, απαιτεί έως και 999 συγκρίσεις. Καθώς το *n* μεγαλώνει, «κυριαρχεί» το *n* μέρος της έκφρασης *n* – 1 και η αφαίρεση του ένα γίνεται ασήμαντη. Το Big *O* έχει σχεδιαστεί να επισημαίνει αυτούς τους κυρίαρχους όρους και να αγνοεί όρους που γίνονται ασήμαντοι καθώς μεγαλώνει το *n*. Για το λόγο αυτό, ένας αλγόριθμος που απαιτεί συνολικά *n* – 1 συγκρίσεις (όπως αυτός που περιγράψαμε νωρίτερα) λέγεται ότι είναι *O*(*n*). Ένας αλγόριθμος **O(n)** αναφέρεται ότι έχει **γραμμικό χρόνο εκτέλεσης**. Το *O*(*n*) προφέρεται «τάξης *n*».

Αλγόριθμοι O(n²)

Τώρα υποθέστε ότι έχετε έναν αλγόριθμο που ελέγχει εάν *κάποιο* στοιχείο ενός πίνακα υπάρχει σε άλλο σημείο του πίνακα. Το πρώτο στοιχείο πρέπει να συγκριθεί με όλα τα άλλα στοιχεία του πίνακα. Το δεύτερο στοιχείο πρέπει να συγκριθεί με όλα τα στοιχεία εκτός από το πρώτο (είχε ήδη συγκριθεί με το πρώτο). Το τρίτο στοιχείο πρέπει να συγκριθεί με όλα τα στοιχεία εκτός από τα δύο πρώτα. Στο τέλος, αυτός ο αλγόριθμος κάνει (*n* - 1) + (*n* - 2) + ... + 2 + 1 ή $n^2/2$ – n/2 συγκρίσεις. Καθώς αυξάνει το *n*, κυριαρχεί ο όρος n^2 και ο όρος *n* γίνεται ασήμαντος. Και πάλι, η σύνταξη Big *O* επισημαίνει τον όρο n^2, αγνοώντας το *n*/2.

Το Big *O* ασχολείται με το πώς αυξάνεται ο χρόνος εκτέλεσης ενός αλγορίθμου σε σχέση με τον αριθμό των στοιχείων που επεξεργάζεται. Ας υποθέσουμε ότι ένας αλγόριθμος απαιτεί n^2 συγκρίσεις. Με τέσσερα στοιχεία, ο αλγόριθμος απαιτεί 16 συγκρίσεις, με οκτώ στοιχεία απαιτεί 64 συγκρίσεις. Με αυτόν τον αλγόριθμο, ο **διπλασιασμός** του αριθμού των στοιχείων **τετραπλασιάζει** τον αριθμό των συγκρίσεων. Φαντασθείτε έναν παρόμοιο αλγόριθμο που απαιτεί $n^2/2$ συγκρίσεις. Με τέσσερα στοιχεία, ο αλγόριθμος απαιτεί οκτώ συγκρίσεις. με οκτώ στοιχεία απαιτεί 32 συγκρίσεις. Και πάλι, ο **διπλασιασμός** του αριθμού των στοιχείων **τετραπλασιάζει** τον αριθμό των συγκρίσεων. Και οι δύο αυτοί αλγόριθμοι αυξάνονται ως προς το τετράγωνο του *n,* οπότε το *Big* O αγνοεί τη σταθερά και οι δύο αλγόριθμοι θεωρούνται **O(n²)**, που αναφέρεται ως **τετραγωνικός χρόνος εκτέλεσης** και προφέρεται «της τάξης του *n*-τετράγωνο» ή πιο απλά «τάξης n-τετράγωνο».

Αποτελεσματικότητα της Γραμμικής Αναζήτησης O(n)

Ο αλγόριθμος γραμμικής αναζήτησης, ο οποίος χρησιμοποιείται συνήθως για αναζήτηση σε έναν μη ταξινομημένο πίνακα, εκτελείται σε *χρόνο* **O(n)**. Η χειρότερη περίπτωση σε αυτόν τον αλγόριθμο είναι όταν πρέπει να ελεγχθεί κάθε στοιχείο για να προσδιοριστεί εάν το κλειδί αναζήτησης υπάρχει στον πίνακα. Εάν το μέγεθος του πίνακα διπλασιαστεί, διπλασιάζεται επίσης και ο αριθμός των συγκρίσεων που πρέπει να εκτελέσει ο αλγόριθμος. Η γραμμική αναζήτηση μπορεί να προσφέρει εξαιρετική απόδοση εάν το στοιχείο που ταιριάζει με το κλειδί αναζήτησης τυχαίνει να βρίσκεται στην αρχή ή κοντά στην αρχή του πίνακα. Ωστόσο, αναζητούμε αλγόριθμους που αποδίδουν καλά, κατά μέσο όρο, σε όλες τις αναζητήσεις, συμπεριλαμβανομένων εκείνων όπου το στοιχείο που ταιριάζει με το κλειδί αναζήτησης βρίσκεται κοντά στο τέλος του πίνακα.

Η γραμμική αναζήτηση είναι εύκολο να προγραμματιστεί, αλλά μπορεί να είναι αργή σε σύγκριση με άλλους αλγόριθμους αναζήτησης, ειδικά καθώς το *n* μεγαλώνει. Εάν ένα πρόγραμμα χρειάζεται να εκτελέσει πολλές αναζητήσεις σε μεγάλους πίνακες, είναι καλύτερο να χρησιμοποιήσετε έναν πιο αποτελεσματικό αλγόριθμο, όπως τον αλγόριθμο `binary_search`, τον οποίο θα χρησιμοποιήσουμε στο Κεφάλαιο 14. Θα δείξουμε πώς να εφαρμόσετε τον αλγόριθμο δυαδικής αναζήτησης στο Κεφάλαιο 21.

Μερικές φορές, οι απλούστεροι αλγόριθμοι έχουν κακή απόδοση. Το πλεονέκτημά τους συνήθως είναι ότι είναι εύκολο να προγραμματιστούν, να δοκιμαστούν και να διορθωθούν. Μερικές φορές απαιτούνται πιο περίπλοκοι αλγόριθμοι για την επίτευξη της μέγιστης απόδοσης.

ΑΠΟΔ

Αποτελεσματικότητα της Δυαδικής Αναζήτησης O(log n)

Στη χειρότερη περίπτωση, η αναζήτηση σε έναν **ταξινομημένο πίνακα** με 1.023 στοιχεία (2^{10} – 1) χρειάζεται **μόνο 10 συγκρίσεις** με την δυαδική αναζήτηση. Ο αλγόριθμος συγκρίνει το κλειδί αναζήτησης με το μεσαίο στοιχείο του ταξινομημένου πίνακα. Εάν υπάρχει ταίριασμα, η

αναζήτηση έχει τελειώσει. Πιθανότατα, το κλειδί αναζήτησης θα είναι μεγαλύτερο ή μικρότερο από το μεσαίο στοιχείο:

- Εάν είναι μεγαλύτερο, μπορούμε να αφαιρέσουμε το πρώτο μισό του πίνακα από την αναζήτηση.
- Εάν είναι μικρότερο, μπορούμε να αφαιρέσουμε το δεύτερο μισό του πίνακα από την αναζήτηση.

Αυτό δημιουργεί ένα **φαινόμενο μείωσης κατά το ήμισυ**, έτσι ώστε στις επόμενες αναζητήσεις, να πρέπει να κάνουμε αναζήτηση μόνο σε 511, 255, 127, 63, 31, 15, 7, 3 και 1 στοιχεία. Ο αριθμός 1.023 (210 – 1) θα χρειασθεί να μειωθεί κατά το ήμισυ μόνο 10 φορές για να βρεθεί το κλειδί ή να προσδιοριστεί ότι δεν βρίσκεται στον πίνακα. Η διαίρεση με το 2 ισοδυναμεί με μια σύγκριση στον αλγόριθμο δυαδικής αναζήτησης. Έτσι, ένας πίνακας 1.048.575 (2^{20} – 1) στοιχείων χρειάζεται το **πολύ 20 συγκρίσεις** για να βρεθεί το κλειδί και ένας πίνακας περίπου ενός δισεκατομμυρίου στοιχείων χρειάζεται **το πολύ 30 συγκρίσεις** για να βρεθεί το κλειδί. Αυτή είναι μια τεράστια βελτίωση στην απόδοση σε σχέση με τη γραμμική αναζήτηση. Για έναν πίνακα ενός δισεκατομμυρίου στοιχείων, αυτή είναι μια διαφορά μεταξύ ενός μέσου όρου 500 εκατομμυρίων συγκρίσεων για τη γραμμική αναζήτηση και ενός μέγιστου μόνο 30 συγκρίσεων για τη δυαδική αναζήτηση! Ο μέγιστος αριθμός συγκρίσεων που απαιτείται από τη δυαδική αναζήτηση οποιουδήποτε ταξινομημένου πίνακα είναι ο εκθέτης της πρώτης δύναμης του 2 μεγαλύτερη από τον αριθμό των στοιχείων του πίνακα, που αντιπροσωπεύεται ως $\log_2 n$. Όλοι οι λογάριθμοι αυξάνονται «περίπου με τον ίδιο ρυθμό», οπότε για σκοπούς σύγκρισης σε σχέση με το Big *O*, η βάση μπορεί να παραλειφθεί. Αυτό έχει ως αποτέλεσμα ένα Big *O* επιπέδου **O(log n)** για μια δυαδική αναζήτηση, η οποία είναι επίσης γνωστή ως **λογαριθμικός χρόνος εκτέλεσης** και προφέρεται ως «τάξης log *n*».

Σύγκριση Συμβολισμών Big *O*

Ο παρακάτω πίνακας παραθέτει διάφορους συνηθισμένους συμβολισμούς Big *O* και διάφορες τιμές για το *n* για να επισημάνει τις διαφορές στους ρυθμούς ανάπτυξης. Αν μεταφράσετε τις τιμές στον πίνακα ως δευτερόλεπτα με υπολογισμούς, μπορείτε εύκολα να δείτε γιατί πρέπει να αποφεύγονται οι αλγόριθμοι $O(n^2)$!

n =	*O*(1)	*O*(log n)	*O*(n)	*O*(n log n)	*O*(n^2)
1	1	0	1	0	1
2	1	1	2	2	4
3	1	1	3	3	9
4	1	1	4	4	16
5	1	1	5	5	25
10	1	1	10	10	100
100	1	2	100	200	10,000
1,000	1	3	1,000	3,000	10^6
1,000,000	1	6	1,000,000	6,000,000	10^{12}

Στην υλοποίηση των κοντέινερ της πρότυπης βιβλιοθήκης (Κεφάλαιο 13) και αλγορίθμων (Κεφάλαιο 14), οι Big *O* αποδόσεις **O(1)**, **O(n)**, **O(log n)** και ακόμη και **O(n log n)**—κάτι που είναι συνηθισμένο για τους σχετικά καλούς αλγόριθμους ταξινόμησης—θεωρούνται αρκετά αποτελεσματικές. Οι αλγόριθμοι που είναι κατηγοριοποιημένοι ως **O(n^2)** ή χειρότεροι, όπως οι **O(2^n)** ή **O(n!)**, θα μπορούσαν να εκτελεστούν σε ένα μέτριο αριθμό στοιχείων για αιώνες, χιλιετίες ή περισσότερο. Επομένως, θα πρέπει να αποφύγετε να γράψετε τέτοιους αλγορίθμους.

13.3 Μια Σύντομη Εισαγωγή στους Πίνακες Κατακερματισμού

Όταν ένα πρόγραμμα δημιουργεί αντικείμενα, ίσως χρειαστεί να τα αποθηκεύσει και να τα ανακτήσει αποτελεσματικά. Η αποθήκευση και η ανάκτηση πληροφοριών με πίνακες είναι αποτελεσματική εάν κάποια πτυχή των δεδομένων σας ταιριάζει κατευθείαν με μια αριθμητική τιμή κλειδιού και εάν τα **κλειδιά είναι μοναδικά** και **πυκνά κατανεμημένα**. Εάν έχετε 100 υπαλλήλους με εννιαψήφιους αριθμούς κοινωνικής ασφάλισης και θέλετε να αποθηκεύσετε και να ανακτήσετε δεδομένα των υπαλλήλων χρησιμοποιώντας τον αριθμό κοινωνικής ασφάλισης ως δείκτη πίνακα, η εργασία θα απαιτήσει έναν πίνακα με πάνω από 800 εκατομμύρια στοιχεία, επειδή οι εννιαψήφιοι αριθμοί κοινωνικής ασφάλισης πρέπει να ξεκινούν με 001-899 (εκτός από το 666) σύμφωνα με τον ιστότοπο του Social Security Administration των ΗΠΑ:

```
https://www.ssa.gov/employer/randomization.html
```

Ένα πρόγραμμα που έχει έναν τόσο μεγάλο πίνακα θα μπορούσε να επιτύχει υψηλή απόδοση τόσο για την αποθήκευση όσο και για την ανάκτηση των εγγραφών των υπαλλήλων χρησιμοποιώντας απλώς τον αριθμό κοινωνικής ασφάλισης ως δείκτη πίνακα. Αυτό δεν είναι πρακτικό για τις περισσότερες εφαρμογές που χρησιμοποιούν ως κλειδιά τους αριθμούς κοινωνικής ασφάλισης.

Πολλές εφαρμογές έχουν αυτό το πρόβλημα – δηλαδή, είτε τα κλειδιά είναι λάθος τύπου (π.χ. όχι θετικοί ακέραιοι που μπορούν να χρησιμοποιηθούν ως δείκτες πίνακα) είτε είναι του σωστού τύπου, αλλά είναι αραιά κατανεμημένοι σε ένα τεράστιο εύρος, όπως οι αριθμοί κοινωνικής ασφάλισης για τους υπαλλήλους μιας μικρής εταιρείας. Αυτό που χρειάζεται είναι ένα σχήμα υψηλής ταχύτητας για τη μετατροπή κλειδιών, όπως είναι οι αριθμοί κοινωνικής ασφάλισης, αριθμοί κωδικών ανταλλακτικών και λοιπά σε μοναδικούς δείκτες πίνακα, σε έναν πίνακα μέτριου μεγέθους. Στη συνέχεια, όταν μια εφαρμογή χρειάζεται να αποθηκεύσει ένα στοιχείο δεδομένων, ο συνδυασμός μπορεί να μετατρέψει γρήγορα το κλειδί της εφαρμογής σε ένα δείκτη πίνακα (μια διαδικασία που ονομάζεται **κατακερματισμός-hashing**) και το στοιχείο μπορεί να αποθηκευτεί σε αυτήν την υποδοχή του πίνακα. Η ανάκτηση επιτυγχάνεται με τον ίδιο τρόπο: Μόλις η εφαρμογή έχει ένα κλειδί για το οποίο θέλει να ανακτήσει τα δεδομένα του, απλώς εφαρμόζει τη μετατροπή στο κλειδί (και πάλι, ονομάζεται **κατακερματισμός**) και αυτό παράγει το δείκτη πίνακα όπου είναι αποθηκευμένα τα δεδομένα και μπορούν να ανακτηθούν.

Από πού προήρθε το όνομα κατακερματισμός; Όταν μετατρέπουμε ένα κλειδί σε δείκτη πίνακα, κυριολεκτικά ανακατεύουμε τα bit, σχηματίζοντας ένα είδος «αλλοιωμένου» ή κατακερματισμένου αριθμού. Ο αριθμός στην πραγματικότητα δεν έχει καμία πραγματική σημασία πέρα από τη χρησιμότητά του στην αποθήκευση και ανάκτηση συγκεκριμένων δεδομένων.

Ένα μειονέκτημα στο σχήμα αυτό ονομάζεται **διένεξη**—αυτό συμβαίνει όταν δύο διαφορετικά κλειδιά «κατακερματίζονται» στο ίδιο κελί (ή στοιχείο) του πίνακα. Δεν μπορούμε να αποθηκεύσουμε δύο τιμές στον ίδιο χώρο, επομένως πρέπει να βρούμε ένα εναλλακτικό σπίτι για όλες τις τιμές πέρα από την πρώτη που καταλήγουν με τον κατακερματισμό στον ίδιο δείκτη πίνακα. Υπάρχουν πολλοί τρόποι να γίνει αυτό. Ο ένας είναι να γίνει «κατακερματισμός ξανά»—δηλαδή, να εφαρμόσετε έναν άλλο μετασχηματισμό κατακερματισμού στο προηγούμενο αποτέλεσμα κατακερματισμού για να βρείτε το επόμενο υποψήφιο κελί στον πίνακα. Η διαδικασία κατακερματισμού έχει σχεδιαστεί ώστε να κατανέμει τις τιμές σε ολόκληρο τον πίνακα, οπότε ελπίζουμε ότι θα βρεθεί ένα διαθέσιμο κελί με έναν ή μερικούς κατακερματισμούς.

Ένα άλλο σχήμα χρησιμοποιεί έναν κατακερματισμό για να εντοπίσει το πρώτο υποψήφιο κελί. Εάν αυτό το κελί είναι κατειλημμένο, γίνεται αναζήτηση διαδοχικά στα γειτονικά κελιά μέχρι να βρεθεί ένα διαθέσιμο κελί. Η ανάκτηση λειτουργεί με τον ίδιο τρόπο: Το κλειδί κατακερματίζεται μια φορά για να προσδιοριστεί η αρχική θέση και να ελεγχθεί εάν περιέχει τα επιθυμητά δεδομένα. Εάν ναι, η αναζήτηση έχει ολοκληρωθεί. Διαφορετικά, γίνεται αναζήτηση σε διαδοχικά κελιά μέχρι να βρεθούν τα επιθυμητά δεδομένα. Μια δημοφιλής λύση για τις διενέξεις κατακερματισμού-πίνακα είναι κάθε κελί του πίνακα να είναι μια **υποδοχή κατακερματισμού**—συνήθως μια συνδεδεμένη λίστα όλων των ζευγαριών κλειδιού-τιμής που κατακερματίζονται σε αυτό το κελί.

Ο **συντελεστής φόρτου** ενός πίνακα κατακερματισμού επηρεάζει την απόδοση των σχημάτων κατακερματισμού. Ο συντελεστής φόρτου είναι ο λόγος του αριθμού των κατειλημμένων κελιών στον πίνακα κατακερματισμού δια του συνολικού αριθμού κελιών στον πίνακα κατακερματισμού. Όσο πιο κοντά πλησιάζει αυτή η αναλογία στο 1,0, τόσο μεγαλύτερη είναι η πιθανότητα διενέξεων, οι οποίες επιβραδύνουν την εισαγωγή και ανάκτηση δεδομένων.

ΑΠΟΔ Ο συντελεστής φόρτου σε έναν πίνακα κατακερματισμού είναι ένα κλασικό παράδειγμα συμβιβασμού μεταξύ **χώρου μνήμης/χρόνου εκτέλεσης**: Αυξάνοντας τον συντελεστή φόρτου, έχουμε καλύτερη χρήση μνήμης, αλλά το πρόγραμμα εκτελείται πιο αργά λόγω αυξημένων διενέξεων κατακερματισμού. Μειώνοντας τον συντελεστή φόρτου, έχουμε καλύτερη ταχύτητα του προγράμματος, λόγω μειωμένων διενέξεων κατακερματισμού, αλλά έχουμε χειρότερη χρήση μνήμης επειδή παραμένει κενό ένα μεγαλύτερο τμήμα του πίνακα κατακερματισμού.

Τα κοντέινερ συσχέτισης **`unordered_set`**, **`unordered_multiset`**, **`unordered_map`** και **`unordered_multimap`** της C++, που θα μελετήσουμε σε αυτό το κεφάλαιο, υλοποιούνται ως πίνακες κατακερματισμού «στο παρασκήνιο». Όταν χρησιμοποιείτε αυτά τα κοντέινερ, έχετε το πλεονέκτημα της αποθήκευσης και ανάκτησης δεδομένων σε υψηλή ταχύτητα χωρίς να χρειάζεται να δημιουργήσετε τους δικούς σας μηχανισμούς πίνακα κατακερματισμού—ένα κλασικό παράδειγμα επαναχρησιμοποίησης.

Μια άλλη ενδιαφέρουσα εφαρμογή του κατακερματισμού είναι σε σχέση με την εικονική μνήμη[3] των λειτουργικών συστημάτων. Προγράμματα και δεδομένα διατηρούνται σε μεγάλο δευτερεύοντα χώρο αποθήκευσης και μεταφέρονται για εκτέλεση στην πιο περιορισμένη κύρια μνήμη (και ακόμη πιο περιορισμένη και ταχύτερη μνήμη cache). Το τέχνασμα για την αποτελεσματική εκτέλεση εφαρμογών είναι να διατηρείται στην κύρια μνήμη (και στη μνήμη cache) το τμήμα του προγράμματος που πρέπει να εκτελείται σε κάθε δεδομένη στιγμή. Η εύρεση αυτών των τμημάτων του προγράμματος στην εικονική μνήμη περιλαμβάνει την αναζήτηση στις πολύ μεγάλες δομές δεδομένων που την ορίζουν. Ο κατακερματισμός χρησιμοποιείται συνήθως για το σκοπό αυτό.

13.4 Εισαγωγή στα Κοντέινερ[4]

Η πρότυπη βιβλιοθήκη των κοντέινερ χωρίζεται σε τέσσερις μεγάλες κατηγορίες:

- κοντέινερ ακολουθίας,
- διατεταγμένα κοντέινερ συσχέτισης,
- μη διατεταγμένα κοντέινερ συσχέτισης και
- προσαρμογείς κοντέινερ.

Συνοψίζουμε εν συντομία τα κοντέινερ εδώ και παρουσιάζουμε πολλά παραδείγματα κώδικα στις ακόλουθες ενότητες.

Κοντέινερ Ακολουθίας

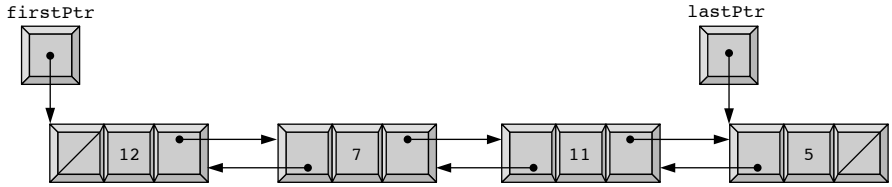
Τα **κοντέινερ ακολουθίας** αντιπροσωπεύουν γραμμικές δομές δεδομένων με όλα τα στοιχεία τους εννοιολογικά «παραταγμένα σε μια σειρά», όπως `array`, `vector` και συνδεδεμένες λίστες, τα οποία μπορούν να ταξινομηθούν. Τα πέντε **κοντέινερ ακολουθίας** είναι:

- **`array`** (Κεφάλαιο 6)—Σταθερό μέγεθος. Τα στοιχεία είναι συνεχόμενα στη μνήμη. **Άμεση πρόσβαση (ονομάζεται επίσης τυχαία πρόσβαση) σε οποιοδήποτε στοιχείο**.
- **`vector`** (Ενότητα 13.8)—Δυνατότητα αλλαγής μεγέθους. Τα στοιχεία είναι συνεχόμενα στη μνήμη. Γρήγορες εισαγωγές και διαγραφές στο πίσω μέρος. Άμεση πρόσβαση σε οποιοδήποτε στοιχείο.
- **`list`** (Ενότητα 13.9)—Διπλή συνδεδεμένη λίστα με δυνατότητα αλλαγής μεγέθους, γρήγορη εισαγωγή και διαγραφή οπουδήποτε. Μια **διπλή συνδεδεμένη λίστα** υποστηρίζει την διάσχιση της λίστας προς τα εμπρός ή προς τα πίσω και συνήθως υλοποιείται με δύο «αρχικούς δείκτες»—έναν που δείχνει στο πρώτο στοιχείο της λίστας και έναν στο

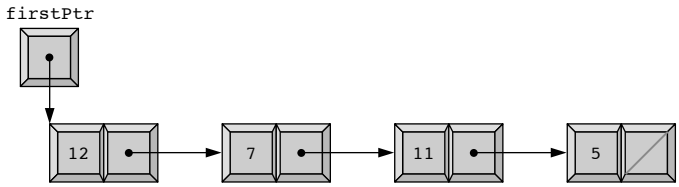
3. "Virtual Memory". Wikipedia. Wikimedia Foundation. Πρόσβαση στις 18 Απριλίου, 2023. `https://en.wikipedia.org/wiki/Virtual_memory`.

4. Αυτή η ενότητα έχει ως στόχο να παρέχει μία εισαγωγή προσανατολισμένη σε αναφορές. Μπορεί να θέλετε να το διαβάσετε γρήγορα και να ανατρέξετε σε αυτό όταν χρειάζεται καθώς διαβάζετε τα παραδείγματα κώδικα του κεφαλαίου.

τελευταίο στοιχείο. Κάθε κόμβος περιέχει δείκτες προς τον προηγούμενο κόμβο και τον επόμενο κόμβο (εξ ου και ο όρος διπλή συνδεδεμένη λίστα), όπως φαίνεται από τα οριζόντια βέλη στο παρακάτω διάγραμμα. Οι κάθετοι (/) στον πρώτο και τον τελευταίο κόμβο αντιπροσωπεύουν κενούς δείκτες (`nullptr`):



- **`deque`** (Ενότητα 13.10)—Δυνατότητα αλλαγής μεγέθους. Γρήγορες εισαγωγές και διαγραφές στο μπροστινό ή πίσω μέρος. Άμεση πρόσβαση σε οποιοδήποτε στοιχείο.
- **`forward_list`**— Μονή συνδεδεμένη λίστα με δυνατότητα αλλαγής μεγέθους, γρήγορη εισαγωγή και διαγραφή οπουδήποτε. Μια **μονή συνδεδεμένη λίστα** διατηρεί συνήθως ένα δείκτη προς τον πρώτο κόμβο της και κάθε κόμβος περιέχει έναν δείκτη προς τον επόμενο κόμβο (εξ ου και ο όρος μονή συνδεδεμένη λίστα), όπως φαίνεται από τα οριζόντια βέλη στο παρακάτω διάγραμμα. Η κάθετος (/) στον τελευταίο κόμβο αντιπροσωπεύει ένα `nullptr`:



Τα κοντέινερ συνδεδεμένων λιστών δεν υποστηρίζουν τυχαία πρόσβαση σε κανένα στοιχείο. Η κλάση **`string`** υποστηρίζει την ίδια λειτουργικότητα με ένα κοντέινερ ακολουθίας, αλλά αποθηκεύει μόνο δεδομένα χαρακτήρων.

Συνειρμικά Κοντέινερ

Τα **κοντέινερ συσχέτισης** είναι μη γραμμικές δομές δεδομένων (συνήθως υλοποιούνται ως δυαδικά δέντρα[5,6]) που συνήθως μπορούν να εντοπίσουν γρήγορα τα στοιχεία που ψάχνουν. Τέτοια κοντέινερ μπορούν να αποθηκεύσουν σύνολα τιμών ή **ζεύγη κλειδιού-τιμής**, στα οποία κάθε κλειδί έχει μια συσχετισμένη τιμή. Για παράδειγμα, ένα πρόγραμμα μπορεί να συσχετίσει κωδικούς υπαλλήλων με αντικείμενα `Employee`. Όπως θα δείτε, ορισμένα κοντέινερ συσχέτισης επιτρέπουν πολλαπλές τιμές για κάθε κλειδί. **Τα κλειδιά στα κοντέινερ συσχέτισης είναι αμετάβλητα**—δεν μπορούν να τροποποιηθούν αν δεν τα αφαιρέσετε πρώτα από το κοντέινερ. Τα τέσσερα **διατεταγμένα κοντέινερ συσχέτισης** είναι:

- **`multiset`** (Ενότητα 13.11.1)—Γρήγορη αναζήτηση, επιτρέπονται διπλότυπα.
- **`set`** (Ενότητα 13.11.2)—Ταχεία αναζήτηση, δεν επιτρέπονται διπλότυπα.
- **`multimap`** (Ενότητα 13.11.3)—Γρήγορη αναζήτηση βάσει κλειδιού, επιτρέπονται διπλότυπα κλειδιά. Απεικόνιση ένα-προς-πολλά.
- **`map`** (Ενότητα 13.11.4)—Γρήγορη αναζήτηση βάσει κλειδιών, δεν επιτρέπονται διπλότυπα κλειδιά. Απεικόνιση ένα-προς-πολλά.

5. "set". Πρόσβαση στις 18 Απριλίου, 2023. `https://en.cppreference.com/w/cpp/container/set`.
6. "map". Πρόσβαση στις 18 Απριλίου, 2023. `https://en.cppreference.com/w/cpp/container/map`.

Τα τέσσερα **μη διατεταγμένα κοντέινερ συσχέτισης** (που υλοποιούνται χρησιμοποιώντας κατακερματισμό[7,8]) είναι

- **`unordered_multiset`**—Γρήγορη αναζήτηση, επιτρέπονται διπλότυπα.
- **`unordered_set`**—Γρήγορη αναζήτηση, δεν επιτρέπονται διπλότυπα.
- **`unordered_multimap`**—Απεικόνιση ένα-προς-πολλά, επιτρέπονται διπλότυπα, γρήγορη αναζήτηση βάσει κλειδιού.
- **`unordered_map`**—Απεικόνιση ένα-προς-ένα, δεν επιτρέπονται διπλότυπα, γρήγορη αναζήτηση βάσει κλειδιού.

Προσαρμογείς Κοντέινερ

Η πρότυπη βιβλιοθήκη χειρίζεται τα **`stack`**, **`queue`** και **`priority_queue`** ως **προσαρμογείς κοντέινερ** που επιτρέπουν σε ένα πρόγραμμα να προβάλλει ένα κοντέινερ ακολουθίας με περιορισμένο τρόπο. Οι τρεις προσαρμογείς κοντέινερ είναι

- **`stack`**—Δομή δεδομένων τελευταίου εισερχομένου, πρώτου εξερχομένου (last-in, first-out - LIFO).
- **`queue`**—Δομή δεδομένων πρώτου εισερχομένου, πρώτου εξερχομένου (first-in, first-out - FIFO).
- **`priority_queue`**—Το στοιχείο υψηλότερης προτεραιότητας είναι πάντα το πρώτο στοιχείο που φεύγει.

Σχεδόν Κοντέινερ

Υπάρχουν και άλλοι τύποι κοντέινερ που θεωρούνται ότι ως «**σχεδόν κοντέινερ**»—ενσωματωμένοι πίνακες (Κεφάλαιο 7), `bitset` (Ενότητα 13.13) για διατήρηση συνόλων τιμών σημαίας, `string` και `valarrays` για εκτέλεση μαθηματικών διανυσματικών πράξεων υψηλής ταχύτητας[9] (δεν πρέπει να συγχέεται με το διανυσματικό κοντέινερ). Αυτοί οι τύποι θεωρούνται ότι είναι σχεδόν κοντέινερ επειδή παρουσιάζουν ορισμένες, αλλά όχι όλες, τις δυνατότητες των **κοντέινερ ακολουθίας** και **συσχέτισης**.

13.4.1 Κοινοί Ένθετοι Τύποι σε Κοντέινερ Ακολουθίας και Συσχέτισης

Ο παρακάτω πίνακας δείχνει τους κοινούς τύπους που ορίζονται μέσα σε κάθε ορισμό κλάσης κοντέινερ ακολουθίας και κοντέινερ συσχέτισης. Αυτοί οι **ένθετοι τύποι** χρησιμοποιούνται σε δηλώσεις μεταβλητών που βασίζονται σε πρότυπα, παραμέτρους σε συναρτήσεις και σε τιμές επιστροφής από συναρτήσεις, όπως θα δείτε σε αυτό το κεφάλαιο και το Κεφάλαιο 14. Για παράδειγμα, το `value_type` σε κάθε κοντέινερ αντιπροσωπεύει πάντα τον τύπο του στοιχείου του κοντέινερ.

---

7. "`unordered_set`". Πρόσβαση στις 18 Απριλίου, 2023. `https://en.cppreference.com/w/cpp/container/unordered_set`.
8. "`unordered_map`". Πρόσβαση στις 18 Απριλίου, 2023. `https://en.cppreference.com/w/cpp/container/unordered_map`.
9. Για επισκόπηση του valarray και των μαθηματικών δυνατοτήτων του, δείτε την τεκμηρίωσή του στο `https://en.cppreference.com/w/cpp/numeric/valarray` και δείτε το άρθρο «std::valarray Class in C++» στο `https://www.geeksforgeeks.org/std-valarray-class-c/`.

Ένθετος τύπος	Περιγραφή
`allocator_type`	Ο τύπος του αντικειμένου που χρησιμοποιείται για την **δέσμευση μνήμης του κοντέινερ**—δεν περιλαμβάνεται στο κοντέινερ **`array`**. Τα κοντέινερ που χρησιμοποιούν δέσμευση μνήμης, παρέχουν το καθένα έναν προεπιλεγμένο τρόπο δέσμευσης, που είναι επαρκής για τους περισσότερους προγραμματιστές. Οι προσαρμοσμένοι τρόποι δέσμευσης μνήμης είναι πέρα από το σκοπό αυτού του βιβλίου.
`value_type`	Ο τύπος των στοιχείων του κοντέινερ.
`reference`	Ο τύπος που χρησιμοποιείται για δήλωση μιας αναφοράς σε ένα στοιχείο κοντέινερ.
`const_reference`	Ο τύπος που χρησιμοποιείται για δήλωση μιας αναφοράς σε ένα `const` στοιχείο κοντέινερ.
`pointer`	Ο τύπος που χρησιμοποιείται για δήλωση ενός δείκτη σε ένα στοιχείο κοντέινερ.
`const_pointer`	Ο τύπος που χρησιμοποιείται για δήλωση ενός δείκτη σε ένα `const` στοιχείο κοντέινερ.
`iterator`	Ένας επαναλήπτης που δείχνει σε ένα στοιχείο κοντέινερ.
`const_iterator`	Ένας επαναλήπτης που δείχνει σε ένα στοιχείο ενός `const` κοντέινερ. Χρησιμοποιείται μόνο για την ανάγνωση στοιχείων και την εκτέλεση `const` λειτουργιών.
`reverse_iterator`	Ένας αντίστροφος επαναλήπτης που δείχνει σε ένα στοιχείο κοντέινερ. Διασχίζει ένα κοντέινερ **από το τέλος έως την αρχή**. Δεν παρέχεται για **`forward_list`**.
`const_reverse_iterator`	Ένας αντίστροφος επαναλήπτης που δείχνει σε ένα στοιχείο κοντέινερ και μπορεί να χρησιμοποιηθεί μόνο για την ανάγνωση στοιχείων και την εκτέλεση **`const` λειτουργιών**. Διασχίζει ένα κοντέινερ αντίστροφα. Δεν παρέχεται για **`forward_list`**.
`difference_type`	Ένας τύπος με πρόσημο που αντιπροσωπεύει τον αριθμό των στοιχείων μεταξύ δύο επαναληπτών που αναφέρονται σε στοιχεία του ίδιου κοντέινερ. Μια τιμή αυτού του τύπου επιστρέφεται από τον υπερφορτωμένο τελεστή μείον (-) ενός επαναλήπτη, εάν υπάρχει και με τη συνάρτηση `std::distance`.
`size_type`	Ο **τύπος χωρίς πρόσημο που χρησιμοποιείται για την καταμέτρηση στοιχείων** ενός κοντέινερ και τη **δημιουργία ευρετηρίου** σε ένα κοντέινερ ακολουθίας τυχαίας πρόσβασης.

13.4.2 Κοινές Συναρτήσεις-Μέλη και Μη Μέλη Ενός Κοντέινερ

Τα περισσότερα κοντέινερ παρέχουν παρόμοια λειτουργικότητα. Πολλές λειτουργίες ισχύουν για όλα τα κοντέινερ και άλλες ισχύουν για συγκεκριμένες κατηγορίες κοντέινερ. Οι παρακάτω πίνακες περιγράφουν τις συνήθως διαθέσιμες συναρτήσεις στα περισσότερα κοντέινερ της πρότυπης βιβλιοθήκης. Πριν χρησιμοποιήσετε οποιοδήποτε κοντέινερ, θα πρέπει να μελετήσετε τις δυνατότητές του. Για μια πλήρη λίστα όλων των συναρτήσεων-μελών των κοντέινερ και ποια κοντέινερ τις υποστηρίζουν, δείτε τον **πίνακα συναρτήσεων-μελών στο `cppreference.com`**[10]. Διάφορες συναρτήσεις-μέλη, τις οποίες δεν παραθέτουμε σε αυτήν την ενότητα, καλύπτονται σε όλο το κεφάλαιο καθώς παρουσιάζουμε διάφορα κοντέινερ ακολουθίας, συσχέτισης και προσαρμογείς.

Ειδικές Συναρτήσεις-Μέλη Κοντέινερ

Ο παρακάτω πίνακας περιγράφει τις ειδικές συναρτήσεις-μέλη που παρέχονται από κάθε κοντέινερ. Επιπλέον, κάθε κλάση κοντέινερ παρέχει συνήθως πολλές υπερφορτωμένες συναρτήσεις δημιουργίας για αρχικοποίηση των κοντέινερ και προσαρμογέων κοντέινερ με διάφορους τρόπους. Για παράδειγμα, κάθε κοντέινερ ακολουθίας και συσχέτισης μπορεί να αρχικοποιηθεί από ένα `initializer_list`.

10. "Container Library—Member Functions Table". Πρόσβαση στις 18 Απριλίου, 2023. `https://en.cppref- erence.com/w/cpp/container`.

Ειδική συνάρτηση-μέλος κοντέινερ	Περιγραφή
προεπιλεγμένη συνάρτηση δημιουργίας	Μια συνάρτηση δημιουργίας που αρχικοποιεί ένα κενό κοντέινερ.
συνάρτηση δημιουργίας αντιγραφής	Μια συνάρτηση δημιουργίας που αρχικοποιεί το κοντέινερ ώστε να είναι αντίγραφο ενός υπάρχοντος κοντέινερ του ίδιου τύπου.
τελεστής `copy=`	Αντιγράφει τα στοιχεία ενός κοντέινερ σε ένα άλλο.
συνάρτηση δημιουργίας μετακίνησης	Μετακινεί τα περιεχόμενα ενός υπάρχοντος κοντέινερ σε ένα νέο κοντέινερ του ίδιου τύπου. Το παλιό κοντέινερ δεν περιέχει πλέον τα δεδομένα. Με αυτόν τον τρόπο αποφεύγεται η επιβάρυνση της αντιγραφής κάθε στοιχείου του υπάρχοντος κοντέινερ.
τελεστής `move=`	Μετακινεί τα περιεχόμενα ενός κοντέινερ σε ένα άλλο του ίδιου τύπου. Το παλιό κοντέινερ δεν περιέχει πλέον τα δεδομένα. Με αυτόν τον τρόπο αποφεύγεται η επιβάρυνση της αντιγραφής κάθε στοιχείου του υπάρχοντος κοντέινερ.
συνάρτηση καταστροφής	Μια συνάρτηση καταστροφής για την διαγραφή των στοιχείων ενός κοντέινερ.

Τελεστές Μη-Μέλη Σχεσιακοί και Ισότητας

Οι τελεστές <, <=, >, >=, == και != υποστηρίζονται από τα περισσότερα κοντέινερ. Στην C++17 και προηγούμενες εκδόσεις, είναι υπερφορτωμένοι ως συναρτήσεις μη-μέλη. Στην C++20, ο μεταγλωττιστής συνθέτει τους τελεστές <, <=, >, >= και != χρησιμοποιώντας τον νέο τριαδικό τελεστή σύγκρισης **<=>** και τον τελεστή ==. Όπως δείξαμε στην Ενότητα 11.7, ο μεταγλωττιστής της C++ μπορεί να χρησιμοποιήσει το <=> για να χειριστεί κάθε σύγκριση σχεσιακή και ισότητας ξαναγράφοντάς την με το <=>. Για παράδειγμα, ο μεταγλωττιστής ξαναγράφει το `x < y` ως

```
(x <=> y) < 0
```

Τα μη ταξινομημένα κοντέινερ συσχέτισης δεν υποστηρίζουν τους τελεστές <, <=, > και >=. Οι σχεσιακοί τελεστές και οι τελεστές ισότητας δεν υποστηρίζονται στα `priority_queue`.

Συναρτήσεις-Μέλη που Επιστρέφουν Επαναλήπτες

Ο παρακάτω πίνακας δείχνει συναρτήσεις-μέλη κοντέινερ που επιστρέφουν επαναλήπτες. Το κοντέινερ **`forward_list`** δεν έχει τις συναρτήσεις-μέλη `rbegin`, `rend`, `crbegin` και `crend`.

Συνάρτηση-μέλος κοντέινερ	Περιγραφή
`begin`	Επιστρέφει ένα `iterator` ή ένα `const_iterator` (ανάλογα με το αν το κοντέινερ είναι `const`) που αναφέρεται στο **πρώτο στοιχείο** του κοντέινερ.
`end`	Επιστρέφει ένα `iterator` ή ένα `const_iterator` (ανάλογα με το αν το κοντέινερ είναι `const`) που αναφέρεται στην **επόμενη θέση μετά το τέλος** του κοντέινερ.
`cbegin`	Επιστρέφει ένα `const_iterator` αναφερόμενο στο **πρώτο στοιχείο** του κοντέινερ.
`cend`	Επιστρέφει ένα `const_iterator` αναφερόμενο στην **επόμενη θέση μετά το τέλος** του κοντέινερ.
`rbegin`	Επιστρέφει ένα `reverse_iterator` ή ένα `const_reverse_iterator` (ανάλογα με το αν το κοντέινερ είναι `const`) που αναφέρεται στο **τελευταίο στοιχείο** του κοντέινερ.

`rend`	Επιστρέφει ένα `reverse_iterator` ή ένα `const_reverse_iterator` (ανάλογα με το αν το κοντέινερ είναι `const`) που αναφέρεται στη **θέση πριν από το πρώτο στοιχείο** του κοντέινερ.
`Crbegin`	Επιστρέφει ένα `const_reverse_iterator` που αναφέρεται στο **τελευταίο στοιχείο** του κοντέινερ.
`crend`	Επιστρέφει ένα `const_reverse_iterator` που αναφέρεται στη **θέση πριν από το πρώτο στοιχείο** του κοντέινερ.

Άλλες Συναρτήσεις-Μέλη

Ο παρακάτω πίνακας παραθέτει διάφορες πρόσθετες συναρτήσεις-μέλη των κοντέινερ. Εάν μια συνάρτηση δεν υποστηρίζεται σε όλα τα κοντέινερ, καθορίζουμε ποια κοντέινερ την υποστηρίζουν και ποια όχι. Για την πλήρη λίστα των συναρτήσεων-μελών ενός κοντέινερ, ανατρέξτε στη σελίδα τεκμηρίωσής τους, στην οποία μπορείτε να αποκτήσετε πρόσβαση από την διεύθυνση

```
https://en.cppreference.com/w/cpp/container
```

Συνάρτηση-μέλος κοντέινερ	Περιγραφή
`clear`	Καταργεί όλα τα στοιχεία από το κοντέινερ. Δεν υποστηρίζεται για **`array`**.
`contains`	Επιστρέφει `true` εάν το καθορισμένο κλειδί υπάρχει στο κοντέινερ, αλλιώς επιστρέφει `false`. Υποστηρίζεται μόνο σε κοντέινερ συσχέτισης.
`Empty`	Επιστρέφει `true` εάν το κοντέινερ είναι κενό. Διαφορετικά, επιστρέφει `false`.
`emplace`	Κατασκευάζει ένα στοιχείο στη θέση στην οποία θα βρίσκεται στο κοντέινερ. Εάν υπάρχει ήδη ένα στοιχείο σε αυτήν τη θέση, κατασκευάζει το αντικείμενο σε μια διαφορετική θέση και, στη συνέχεια, το μετακινεί στη θέση του χρησιμοποιώντας μετακίνηση με εκχώρηση. Δεν υποστηρίζεται σε **`array`**. Σε μια **`forward_list`** υποστηρίζει τις **`emplace_after`** και **`emplace_front`** και όχι την **`erase`**. Τα κοντέινερ συσχέτισης παρέχουν τις **`emplace`** και **`emplace_hint`**.
`erase`	Διαγράφει ένα ή περισσότερα στοιχεία από το κοντέινερ. Δεν υποστηρίζεται σε **`array`**. Μια **`forward_list`** υποστηρίζει την **`erase_after`** και όχι την **`erase`**.
`extract`	Τα κοντέινερ συσχέτισης είναι **συνδεδεμένες δομές δεδομένων κόμβων** που περιέχουν τιμές. Η συνάρτηση-μέλος `extract` των κοντέινερ συσχέτισης καταργεί έναν κόμβο από το κοντέινερ και επιστρέφει ένα αντικείμενο `node_type` αυτού του κοντέινερ.
`insert`	Εισάγει ένα στοιχείο στο κοντέινερ. Έχει υπερφορτωθεί ώστε να υποστηρίζει **αντιγραφή και μετακίνηση**. Δεν υποστηρίζεται σε **`array`**. Μια **`forward_list`** υποστηρίζει την **`insert_after`** όχι την **`insert`**.
`max_size`	Επιστρέφει το μέγιστο αριθμό στοιχείων ενός κοντέινερ.
`size`	Επιστρέφει τον αριθμό των στοιχείων που υπάρχουν αυτήν τη στιγμή στο κοντέινερ. Δεν υποστηρίζεται σε **`forward_list`**.
`swap`	Εναλλάσσει τα περιεχόμενα δύο κοντέινερ.

13.4.3 Απαιτήσεις για Στοιχεία Κοντέινερ

Πριν χρησιμοποιήσετε ένα κοντέινερ της πρότυπης βιβλιοθήκης, είναι σημαντικό να βεβαιωθείτε ότι ο τύπος του στοιχείου υποστηρίζει ένα ελάχιστο σύνολο λειτουργιών. Για παράδειγμα:

- Εάν η εισαγωγή ενός στοιχείου σε έναν κοντέινερ απαιτεί ένα **αντίγραφο** του αντικειμένου, ο τύπος του αντικειμένου πρέπει να παρέχει μια **συνάρτηση δημιουργίας αντιγραφής** και έναν **τελεστή εκχώρησης με αντιγραφή**.
- Εάν η εισαγωγή ενός στοιχείου σε ένα κοντέινερ απαιτεί **μετακίνηση** του αντικειμένου, ο τύπος του αντικειμένου θα πρέπει να παρέχει μια **συνάρτηση δημιουργίας μετακίνησης** και έναν **τελεστή εκχώρησης με μετακίνηση**—το Κεφάλαιο 11 συζήτησε τη **σημασιολογία της μετακίνησης**.
- Τα **διατεταγμένα κοντέινερ συσχέτισης** και πολλοί αλγόριθμοι απαιτούν τη σύγκριση στοιχείων. Για το λόγο αυτό, ο τύπος του αντικειμένου πρέπει να υποστηρίζει συγκρίσεις.

Καθώς επισκοπείτε την τεκμηρίωση κάθε κοντέινερ, είτε στο ίδιο το πρότυπο έγγραφο της C++ είτε σε δικτυακούς τόπους όπως το `cppreference.com`, θα δείτε διάφορες **καθορισμένες απαιτήσεις**, όπως τις **CopyConstructible**, **MoveAssignable** ή **EqualityComparable**. Αυτές σας βοηθούν να προσδιορίσετε εάν οι τύποι σας είναι συμβατοί με διάφορα κοντέινερ και αλγόριθμους της πρότυπης βιβλιοθήκης της C++. Μπορείτε να προβάλετε μια λίστα με τις καθορισμένες απαιτήσεις και τις περιγραφές τους από τη διεύθυνση

```
https://en.cppreference.com/w/cpp/named_req
```

Στην C++20, πολλές καθορισμένες απαιτήσεις επισημοποιούνται ως **έννοιες** (concept), τις οποίες θα συζητήσουμε στα Κεφάλαια 14 και 15.

13.5 Εργασία με Επαναλήπτες

Οι **επαναλήπτες** (iterators) έχουν πολλές ομοιότητες με τους δείκτες. Δείχνουν σε στοιχεία σε κοντέινερ ακολουθίας και κοντέινερ συσχέτισης, αλλά **διατηρούν επίσης πληροφορίες κατάστασης σε σχέση με τα συγκεκριμένα κοντέινερ στα οποία λειτουργούν**. Έτσι, υπάρχει διαφορετικός χειρισμός των επαναληπτών για κάθε τύπο κοντέινερ. Η σύνταξη μιας λειτουργίας ενός επαναλήπτη είναι ομοιόμορφη σε όλα τα κοντέινερ. Για παράδειγμα, η εφαρμογή του `*` σε έναν επαναλήπτη καταργεί την αναφορά του, ώστε να έχετε πρόσβαση στο αναφερόμενο στοιχείο. Η εφαρμογή του `++` σε έναν επαναλήπτη τον μετακινεί στο επόμενο στοιχείο του κοντέινερ. Αυτή η ομοιομορφία είναι μια βασική πτυχή των επαναληπτών, επιτρέποντας στους αλγόριθμους της πρότυπης βιβλιοθήκης να λειτουργούν σε διάφορους τύπους κοντέινερ χρησιμοποιώντας πολυμορφισμό κατά τη μεταγλώττιση.

Τα κοντέινερ ακολουθίας και τα κοντέινερ συσχέτισης παρέχουν τις συναρτήσεις-μέλη `begin` και `end`. Η συνάρτηση **`begin`** επιστρέφει έναν επαναλήπτη που δείχνει στο πρώτο στοιχείο του κοντέινερ. Η συνάρτηση **`end`** επιστρέφει έναν επαναλήπτη που δείχνει στο **πρώτο στοιχείο μετά το τέλος του κοντέινερ** (ένα μετά το τέλος)—ένα ανύπαρκτο στοιχείο που χρησιμοποιείται συχνά για να προσδιοριστεί αν έχει φτάσει το τέλος ενός κοντέινερ. Συνήθως θα την χρησιμοποιείτε σε συγκρίσεις ισότητας ή ανισότητας για να προσδιορίσετε εάν ένας επαναλήπτης που αυξάνεται έχει φτάσει στο τέλος του κοντέινερ.

13.5.1 Χρήση του `istream_iterator` για Είσοδο και του `ostream_iterator` για Έξοδο

Χρησιμοποιούμε επαναλήπτες με **ακολουθίες** (ονομάζονται επίσης **εύρη**). Αυτοί μπορεί να είναι σε κοντέινερ ή μπορεί να είναι **ακολουθίες εισόδου** ή **ακολουθίες εξόδου**. Η Εικόνα 13.1 δείχνει

- είσοδο από την τυπική είσοδο (μια ακολουθία δεδομένων για είσοδο σε ένα πρόγραμμα), χρησιμοποιώντας ένα **`istream_iterator`**, και
- έξοδο στην τυπική έξοδο (μια ακολουθία δεδομένων για έξοδο από ένα πρόγραμμα), χρησιμοποιώντας ένα **`ostream_iterator`**.

Το πρόγραμμα λαμβάνει δύο ακέραιους αριθμούς από το χρήστη και εμφανίζει το άθροισμά τους. Όπως θα δείτε αργότερα σε αυτό το κεφάλαιο, τα `istream_iterator` και `ostream_iterator` μπορούν να χρησιμοποιηθούν με τους αλγόριθμους της πρότυπης βιβλιοθήκης για δημιουργία ισχυρών προτάσεων. Για παράδειγμα, επόμενα παραδείγματα θα χρησιμοποιήσουν ένα `ostream_iterator` με τον αλγόριθμο `copy` για να αντιγράψουν τα στοιχεία ενός κοντέινερ στην τυπική ροή εξόδου με μια μόνο πρόταση.

```
1  // fig13_01.cpp
2  // Επίδειξη εισόδου και εξόδου με επαναλήπτες.
3  #include <iostream>
4  #include <iterator> // ostream_iterator και istream_iterator
5
6  int main() {
7     std::cout << "Enter two integers: ";
8
9     // δημιουργία istream_iterator για ανάγνωση int τιμών από το cin
10    std::istream_iterator<int> inputInt{std::cin};
11
12    const int number1{*inputInt}; // ανάγνωση ακέραιου από την τυπική είσοδο
13    ++inputInt; // μετακίνηση επαναλήπτη στην επόμενη τιμή εισόδου
14    const int number2{*inputInt}; // ανάγνωση ακέραιου από την τυπική είσοδο
15
16    // δημιουργία ostream_iterator για γράψιμο int τιμών στο cout
17    std::ostream_iterator<int> outputInt{std::cout};
18
19    std::cout << "The sum is: ";
20    *outputInt = number1 + number2; // αποτέλεσμα εξόδου στο cout
21    std::cout << "\n";
22 }
```

```
Enter two integers: 12 25
The sum is: 37
```

Εικόνα 13.1 | Επίδειξη εισόδου και εξόδου με επαναλήπτες.

`istream_iterator`
Η γραμμή 10 δημιουργεί ένα `istream_iterator` ικανό να **εξάγει** (**εισάγει**) `int` τιμές από το τυπικό αντικείμενο εισόδου `cin`. Η γραμμή 12 **καταργεί την αναφορά** του επαναλήπτη `inputInt` για να διαβάσει τον πρώτο ακέραιο από το `cin` και αρχικοποιεί το `number1` με την τιμή αυτή. Ο τελεστής διακοπής αναφοράς `*` που εφαρμόζεται στο `inputInt` λαμβάνει την τιμή από τη ροή. Η γραμμή 13 τοποθετεί το `inputInt` στην επόμενη τιμή στη ροή εισόδου. Η γραμμή 14 εισάγει τον επόμενο `int` από το `inputInt` και αρχικοποιεί το `number2` με αυτόν. Μπορεί να χρησιμοποιηθεί προσαύξηση είτε προθέματος είτε επιθέματος. Χρησιμοποιούμε τη μορφή προθέματος για λόγους απόδοσης, επειδή δεν δημιουργεί ένα προσωρινό αντικείμενο.

`ostream_iterator`
Η γραμμή 17 δημιουργεί έναν `ostream_iterator` ικανό να **εισάγει** (εξάγει) `int` τιμές στο αντικείμενο της τυπικής εξόδου `cout`. Η γραμμή 20 εξάγει έναν ακέραιο στο `cout` εκχωρώντας στον επαναλήπτη με διακοπή αναφοράς (`*outputInt`) το άθροισμα των `number1` και `number2`.

13.5.2 Κατηγορίες Επαναληπτών

Ο παρακάτω πίνακας περιγράφει τις κατηγορίες επαναληπτών. Η καθεμία παρέχει ένα συγκεκριμένο σύνολο λειτουργιών. Σε όλο αυτό το κεφάλαιο, συζητάμε ποια κατηγορία επαναλήπτη υποστηρίζει κάθε κοντέινερ. Στο Κεφάλαιο 14, θα δείτε ότι οι ελάχιστες απαιτήσεις ενός αλγορίθμου για έναν επαναλήπτη προσδιορίζουν ποια κοντέινερ μπορούν να χρησιμοποιηθούν με αυτόν τον αλγόριθμο.

Κατηγορία επαναλήπτη	Περιγραφή
είσοδος (input)	Χρησιμοποιείται για την ανάγνωση ενός στοιχείου από ένα κοντέινερ. Μπορεί να μετακινηθεί μόνο προς τα εμπρός κατά ένα στοιχείο κάθε φορά, από την αρχή του κοντέινερ έως το τέλος του. **Οι επαναλήπτες εισόδου υποστηρίζουν μόνο αλγόριθμους ενός περάσματος. Ο ίδιος επαναλήπτης εισόδου δεν μπορεί να χρησιμοποιηθεί για να περάσει από μια ακολουθία δύο φορές.**
έξοδος (output)	Χρησιμοποιείται για την εγγραφή ενός στοιχείου σε ένα κοντέινερ. Μπορεί να μετακινηθεί προς τα εμπρός μόνο κατά ένα στοιχείο κάθε φορά. **Οι επαναλήπτες εξόδου υποστηρίζουν μόνο αλγόριθμους ενός περάσματος. Ο ίδιος επαναλήπτης εξόδου δεν μπορεί να χρησιμοποιηθεί για να περάσει από μια ακολουθία δύο φορές.** Για τους επόμενους τύπους επαναληπτών σε αυτόν τον πίνακα, εάν αναφέρονται σε μη σταθερά δεδομένα, οι επαναλήπτες μπορούν να χρησιμοποιηθούν για εγγραφή στο κοντέινερ.
προώθησης (forward)	Έχει τις δυνατότητες **επαναληπτών εισόδου και εξόδου** και διατηρεί τη θέση του στο κοντέινερ. **Τέτοιοι επαναλήπτες μπορούν να περάσουν από μια ακολουθία περισσότερες από μια φορές για αλγόριθμους πολλαπλής διέλευσης.**
αμφίδρομο (bidirectional)	Έχει τις δυνατότητες ενός **επαναλήπτη προώθησης** και επιπλέον έχει τη δυνατότητα να κινείται προς τα πίσω από το τέλος του κοντέινερ προς την αρχή του. Οι αμφίδρομοι επαναλήπτες υποστηρίζουν αλγόριθμους πολλαπλής διέλευσης.
τυχαίας προσπέλασης (random access)	Έχει τις δυνατότητες ενός **αμφίδρομου επαναλήπτη** και επιπλέον έχει τη δυνατότητα άμεσης πρόσβασης σε οποιοδήποτε στοιχείο του κοντέινερ—δηλαδή, μεταπήδηση προς τα εμπρός ή προς τα πίσω κατά έναν αυθαίρετο αριθμό στοιχείων. **Αυτοί μπορούν επίσης να συγκριθούν με τους σχεσιακούς τελεστές.**
συνεχόμενο (contiguous)	Ένας **επαναλήπτης τυχαίας προσπέλασης** που απαιτεί την αποθήκευση στοιχείων σε συνεχόμενες θέσεις της μνήμης.

13.5.3 Υποστήριξη των Κοντέινερ σε Επαναλήπτες

Η κατηγορία επαναληπτών που υποστηρίζει κάθε κοντέινερ προσδιορίζει εάν αυτό το κοντέινερ μπορεί να χρησιμοποιηθεί με συγκεκριμένους αλγόριθμους. **Τα κοντέινερ που υποστηρίζουν επαναλήπτες τυχαίας προσπέλασης μπορούν να χρησιμοποιηθούν με όλους τους αλγόριθμους της πρότυπης βιβλιοθήκης.** Μπορούν να χρησιμοποιηθούν δείκτες ως επαναλήπτες σε ενσωματωμένους πίνακες. Ο παρακάτω πίνακας δείχνει την κατηγορία επαναληπτών κάθε κοντέινερ. Μπορείτε να διασχίσετε με επαναλήπτες κοντέινερ ακολουθίας, κοντέινερ συσχέτισης, συμβολοσειρές και ενσωματωμένους πίνακες.

Κοντέινερ	Τύπος επαναλήπτη	Κοντέινερ	Τύπος επαναλήπτη
Κοντέινερ ακολουθίας		**Διατεταγμένα κοντέινερ συσχέτισης**	
`vector`	συνεχόμενο	`set`	αμφίδρομος
`array`	συνεχόμενο	`multiset`	αμφίδρομος

deque	τυχαίας προσπέλασης	**map**	αμφίδρομος
list	αμφίδρομος	**multimap**	αμφίδρομος
forward_list	προώθησης		
Μη διατεταγμένα κοντέινερ συσχέτισης		**Προσαρμογείς κοντέινερ**	
unordered_set	προώθησης	**stack**	κανένας
unordered_multiset	προώθησης	**queue**	κανένας
unordered_map	προώθησης	**priority_queue**	κανένας
unordered_multimap	προώθησης		

13.5.4 Προκαθορισμένα Ονόματα Τύπων Επαναληπτών

Ο παρακάτω πίνακας εμφανίζει τα προκαθορισμένα ονόματα τύπων επαναληπτών που βρίσκονται στους ορισμούς κλάσεων κοντέινερ της πρότυπης βιβλιοθήκης. Δεν ορίζεται κάθε όνομα τύπου επαναλήπτη για κάθε κοντέινερ. **Οι `const` επαναλήπτες χρησιμοποιούνται για τη διάσχιση κοντέινερ που δεν πρέπει να τροποποιηθούν**. Οι **αντίστροφοι επαναλήπτες** διασχίζουν τα κοντέινερ στην αντίστροφη κατεύθυνση.

Προκαθορισμένο όνομα τύπου επαναλήπτη	Κατεύθυνση του ++	Δυνατότητα
`iterator`	προώθησης	ανάγνωση/εγγραφή
`const_iterator`	προώθησης	ανάγνωση
`reverse_iterator`	αντίστροφος	ανάγνωση/εγγραφή
`const_reverse_iterator`	αντίστροφος	ανάγνωση

Οι λειτουργίες που εκτελούνται με έναν `const_iterator` επιστρέφουν αναφορές σε `const` για να αποτρέψουν την τροποποίηση στοιχείων του κοντέινερ. Η χρήση ενός `const_iterator` όπου ενδείκνυται είναι ένα άλλο παράδειγμα της αρχής των ελάχιστων δικαιωμάτων.

13.5.5 Τελεστές Επαναληπτών

Ο παρακάτω πίνακας δείχνει τους τελεστές για κάθε τύπο επαναλήπτη. Εκτός από τους τελεστές που φαίνονται σε όλους τους επαναλήπτες, οι επαναλήπτες πρέπει να παρέχουν **προεπιλεγμένες συναρτήσεις δημιουργίας** (επαναλήπτες προώθησης και υψηλότεροι), **συναρτήσεις δημιουργίας αντιγραφής** και **τελεστές εκχώρησης με αντιγραφή**. Ένας **επαναλήπτης προώθησης** υποστηρίζει το ++ και όλες τις δυνατότητες **ενός επαναλήπτη εισόδου και εξόδου**. Ένας **αμφίδρομος** επαναλήπτης υποστηρίζει το -- και όλες τις δυνατότητες **ενός επαναλήπτη** προώθησης. Οι **επαναλήπτες τυχαίας προσπέλασης** και οι **συνεχόμενοι επαναλήπτες** υποστηρίζουν όλες τις λειτουργίες που φαίνονται στον πίνακα. Για τους επαναλήπτες εισόδου και τους επαναλήπτες εξόδου, δεν είναι δυνατό να αποθηκεύσετε τον επαναλήπτη και να χρησιμοποιήσετε την αποθηκευμένη τιμή αργότερα.

Πράξη επαναλήπτη	Περιγραφή
Όλοι οι επαναλήπτες	
`++p`	Προ-αύξηση ενός επαναλήπτη, μετακίνησή του στο επόμενο στοιχείο.
`p++`	Μετά-αύξηση ενός επαναλήπτη, μετακίνησή του στο επόμενο στοιχείο.
`p = p1`	Εκχώρηση ενός επαναλήπτη σε έναν άλλο.

Επαναλήπτες εισόδου	
`*p`	Διακοπή αναφοράς ενός επαναλήπτη ως μια `const` *lvalue*.
`p->m`	Χρήση του επαναλήπτη για ανάγνωση του στοιχείου `m`.
`p == p1`	Σύγκριση επαναληπτών για ισότητα.
`p != p1`	Σύγκριση επαναληπτών για ανισότητα.
Επαναλήπτες εξόδου	
`*p`	Διακοπή αναφοράς ενός επαναλήπτη ως μια *lvalue*.
`p = p1`	Εκχώρηση ενός επαναλήπτη σε έναν άλλο.
Επαναλήπτες προώθησης	Οι επαναλήπτες προώθησης παρέχουν όλη τη λειτουργικότητα τόσο των επαναληπτών εισόδου όσο και των επαναληπτών εξόδου.
Αμφίδρομοι επαναλήπτες	
`--p`	Προ-μείωση ενός επαναλήπτη, μετακίνησή του στο επόμενο στοιχείο.
`p--`	Μετά-μείωση ενός επαναλήπτη, μετακίνησή του στο επόμενο στοιχείο.
Επαναλήπτες τυχαίας προσπέλασης	
`p += i`	Αύξηση του επαναλήπτη `p` κατά `i` θέσεις.
`p -= i`	Μείωση του επαναλήπτη `p` κατά `i` θέσεις.
`p + i` ή $i + p$	Η τιμή της έκφρασης είναι ένας επαναλήπτης τοποθετημένος στο `p` αυξημένο κατά `i` θέσεις.
`p - i`	Η τιμή της έκφρασης είναι ένας επαναλήπτης τοποθετημένος στο `p` μειωμένο κατά `i` θέσεις.
`p - p1`	Η τιμή της έκφρασης είναι ένας ακέραιος που αντιπροσωπεύει την απόσταση (δηλαδή, τον αριθμό των στοιχείων) μεταξύ δύο στοιχείων του ίδιου κοντέινερ.
`p[i]`	Επιστροφή αναφοράς στο στοιχείο μετατοπισμένο από το `p` κατά `i` θέσεις
`p < p1`	Επιστρέφει `true` αν ο επαναλήπτης `p` είναι μικρότερος από τον επαναλήπτη `p1` (δηλ. ο επαναλήπτης `p` είναι πριν από τον επαναλήπτη `p1` στο κοντέινερ). Διαφορετικά, επιστρέφει `false`.
`p <= p1`	Επιστρέφει `true` αν ο επαναλήπτης `p` είναι μικρότερος ή ίσος με τον επαναλήπτη `p1` (δηλ. ο επαναλήπτης `p` είναι πριν από τον επαναλήπτη `p1` ή είναι στην ίδια θέση με τον επαναλήπτη `p1` στο κοντέινερ). Διαφορετικά, επιστρέφει `false`.
`p > p1`	Επιστρέφει `true` αν ο επαναλήπτης `p` είναι μεγαλύτερος από τον επαναλήπτη `p1` (δηλ. ο επαναλήπτης `p` είναι μετά τον επαναλήπτη `p1` στο κοντέινερ). Διαφορετικά, επιστρέφει `false`.
`p >= p1`	Επιστρέφει `true` εάν ο επαναλήπτης `p` είναι μεγαλύτερος ή ίσος από τον επαναλήπτη `p1` (δηλαδή, ο επαναλήπτης `p` είναι μετά τον επαναλήπτη `p1` ή στην ίδια θέση με τον επαναλήπτη `p1` στο κοντέινερ). Διαφορετικά, επιστρέφει `false`.

13.6 Μια Σύντομη Εισαγωγή στους Αλγορίθμους

Η πρότυπη βιβλιοθήκη παρέχει δεκάδες αλγορίθμους που μπορείτε να χρησιμοποιήσετε για να χειριστείτε μια μεγάλη ποικιλία κοντέινερ. Η εισαγωγή, διαγραφή, αναζήτηση, ταξινόμηση και λοιπά είναι κατάλληλες για ορισμένα ή για όλα τα κοντέινερ ακολουθίας και συσχέτισης. **Οι αλγόριθμοι λειτουργούν σε στοιχεία κοντέινερ μόνο έμμεσα μέσω επαναληπτών.** Πολλοί αλγόριθμοι λειτουργούν σε ακολουθίες στοιχείων που ορίζονται από επαναλήπτες που δείχνουν προς το **πρώτο στοιχείο** της ακολουθίας και **προς ένα στοιχείο μετά το τελευταίο στοιχείο** – γνωστά ως **μισάνοιχτα εύρη** (half-open ranges). Πολλοί αλγόριθμοι υποστηρίζουν τώρα εύρη στην C++20.

Είναι επίσης δυνατό να δημιουργήσετε τους δικούς σας νέους αλγόριθμους που λειτουργούν με παρόμοιο τρόπο, ώστε να μπορούν να χρησιμοποιηθούν με τα κοντέινερ και επαναλήπτες της πρότυπης βιβλιοθήκης. Σε αυτό το κεφάλαιο, θα χρησιμοποιήσουμε τον αλγόριθμο αντιγραφής σε πολλά παραδείγματα για να αντιγράψουμε τα περιεχόμενα ενός κοντέινερ στην τυπική έξοδο. Συζητάμε πολλούς αλγόριθμους της πρότυπης βιβλιοθήκης στο Κεφάλαιο 14.

13.7 Κοντέινερ Ακολουθίας

Η πρότυπη βιβλιοθήκη της C++ παρέχει πέντε **κοντέινερ ακολουθίας**—**`array`**, **`vector`**, **`deque`**, **`list`** και **`forward_list`**. Τα κοντέινερ `array`, `vector` και `deque` βασίζονται συνήθως σε ενσωματωμένους πίνακες. Τα κοντέινερ `list` και `forward_list` χειρίζονται δομές δεδομένων συνδεδεμένης λίστας. Έχουμε ήδη συζητήσει και χρησιμοποιήσει εκτενώς τα `array` στο Κεφάλαιο 6, οπότε δεν τα καλύπτουμε ξανά εδώ. Έχουμε επίσης παρουσιάσει ήδη το `vector` στο Κεφάλαιο 6 – και το συζητάμε με περισσότερες λεπτομέρειες εδώ.

Απόδοση και Επιλογή του Κατάλληλου Κοντέινερ

Η Ενότητα 13.4.2 παρουσίασε τις λειτουργίες που είναι κοινές στα περισσότερα κοντέινερ της πρότυπης βιβλιοθήκης. Πέρα από αυτές τις λειτουργίες, κάθε κοντέινερ παρέχει συνήθως διάφορες άλλες δυνατότητες. Πολλές είναι κοινές σε πολλά κοντέινερ, αλλά δεν είναι πάντα εξίσου αποτελεσματικές. Οι C++ Core Guidelines λένε ότι το **`vector` είναι ικανοποιητικό για τις περισσότερες εφαρμογές**[11]. Ωστόσο, θα πρέπει να διαμορφώσετε το προφίλ της εφαρμογής σας ώστε να είστε σίγουροι ότι το κοντέινερ που επιλέγετε είναι το σωστό για τις δικές σας απαιτήσεις χρήσης και απόδοσης.

Η εισαγωγή στο τέλος ενός `vector` είναι αποτελεσματική. Το `vector` μεγαλώνει, αν χρειάζεται, για να χωρέσει το νέο στοιχείο. Είναι ακριβό να εισαγάγετε (ή να διαγράφετε) ένα στοιχείο στη μέση ενός `vector`. Κάθε στοιχείο μετά το σημείο εισαγωγής (ή διαγραφής) πρέπει να μετακινηθεί επειδή **τα στοιχεία ενός `vector` καταλαμβάνουν συνεχόμενα κελιά στη μνήμη**.

Σε εφαρμογές που απαιτούν συχνές εισαγωγές και διαγραφές και στα δύο άκρα ενός κοντέινερ, σκεφτείτε να χρησιμοποιήσετε ένα **`deque`** και όχι ένα `vector`. Η κλάση `deque` είναι *O(1)* για εισαγωγές και διαγραφές στην αρχή[12], καθιστώντας την πιο αποτελεσματική από τα `vector`[13].

Εφαρμογές με συχνές εισαγωγές και διαγραφές στη μέση ή/και στα άκρα ενός κοντέινερ χρησιμοποιούν μερικές φορές μια **`list`** λόγω του αποτελεσματικού χειρισμού των εισαγωγών και διαγραφών οπουδήποτε στη δομή δεδομένων.

13.8 Κοντέινερ Ακολουθίας `vector`

Το κοντέινερ **`vector`** (που παρουσιάσθηκε στην Ενότητα 6.15) είναι μια δυναμική δομή δεδομένων που καταλαμβάνει συνεχόμενες θέσεις στη μνήμη. Παρέχει γρήγορη πρόσβαση μέσω δείκτη με τον υπερφορτωμένο τελεστή δείκτη `[]` όπως σε έναν ενσωματωμένο πίνακα ή ένα αντικείμενο `array`. **Επιλέξτε το κοντέινερ `vector` για να έχετε την καλύτερη απόδοση τυχαίας προσπέλασης σε ένα κοντέινερ που μπορεί να μεγαλώσει.** Όταν η χωρητικότητας ενός `vector` ξεπερνιέται,

- **δεσμεύει** έναν μεγαλύτερο ενσωματωμένο πίνακα,

---

11. C++ Core Guidelines, "SL.con.2: Prefer Using STL `vector` By Default Unless You Have a Reason to Use a Different Container". Πρόσβαση στις 18 Απριλίου, 2023. `https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rsl-vector`.

12. «`std::deque`». Πρόσβαση στις 18 Απριλίου, 2023. `https://en.cppreference.com/w/cpp/container/deque`.

13. «`std::vector`.» Πρόσβαση στις 18 Απριλίου, 2023. `https://en.cppreference.com/w/cpp/container/vector`.

- **αντιγράφει** ή **μετακινεί** (ανάλογα με το τι υποστηρίζει ο τύπος του στοιχείου) τα αρχικά στοιχεία στον νέο ενσωματωμένο πίνακα και
- **αποδεσμεύει** τον προηγούμενο ενσωματωμένο πίνακα.

13.8.1 Χρήση `vector` και `iterator`

Η Εικόνα 13.2 απεικονίζει διάφορες συναρτήσεις-μέλη ενός `vector`, πολλές από τις οποίες είναι διαθέσιμες σε κάθε κοντέινερ ακολουθίας και κοντέινερ συσχέτισης. Καθώς προσθέτουμε στοιχεία σε ένα `vector<int>` σε αυτό το παράδειγμα, καλούμε τη συνάρτηση `showResult` (γραμμές 9–12) για να εμφανίσουμε τη νέα τιμή καθώς και το `vector`

- μέγεθος—τον αριθμό των στοιχείων που περιέχει αυτήν τη στιγμή και
- **χωρητικότητα**—τον αριθμό των στοιχείων που μπορεί να αποθηκεύσει προτού χρειαστεί να **αλλάξει δυναμικά το μέγεθός του** για να χωρέσει περισσότερα στοιχεία.

```
1  // fig13_02.cpp
2  // Πρότυπο κλάσης vector της πρότυπης βιβλιοθήκης.
3  #include <format> //
4  #include <iostream>
5  #include <ranges>
6  #include <vector> // ορισμός προτύπου κλάσης vector
7
8  // εμφανίζει την τιμή που έχει προστεθεί και το ενημερωμένο μέγεθος και χωρητικότητα του vector
9  void showResult(int value, size_t size, size_t capacity) {
10    std::cout << std::format("appended: {}; size: {}; capacity: {}\n",
11       value, size, capacity);
12 }
13
```

Εικόνα 13.2 ¦ Πρότυπο κλάσης `vector` της πρότυπης βιβλιοθήκης.

Δημιουργία ενός `vector` και Εμφάνιση του Αρχικού Μεγέθους και Χωρητικότητάς του

Η γραμμή 15 ορίζει το αντικείμενο `vector<int>` με ακέραιους. Η προεπιλεγμένη συνάρτηση δημιουργίας του `vector` δημιουργεί ένα κενό `vector` με μέγεθος και χωρητικότητα ίση με 0. Έτσι, το `vector` θα πρέπει να δεσμεύει μνήμη όταν προστίθενται στοιχεία σε αυτό. Οι γραμμές 17–18 εμφανίζουν το μέγεθος (`size`) και τη χωρητικότητά του (`capacity`). Η συνάρτηση `size` επιστρέφει τον αριθμό των στοιχείων που είναι αποθηκευμένα αυτήν τη στιγμή στο κοντέινερ[14]. Η συνάρτηση **`capacity`** επιστρέφει την τρέχουσα χωρητικότητα του `vector`.

```
14 int main() {
15    std::vector<int> integers{}; // δημιουργία vector από int
16
17    std::cout << "Size of integers: " << integers.size()
18       << "\nCapacity of integers: " << integers.capacity() << "\n\n";
19
```

```
Size of integers: 0
Capacity of integers: 0
```

14. Η `forward_list` δεν έχει τη συνάρτηση-μέλος `size`.