STL
Standard Template Library

Τμήμα Πληροφορικής και Τηλεπικοινωνιών (Άρτα)

Πανεπιστήμιο Ιωαννίνων

Γκόγκος Χρήστος

V 1.0

Standard Template Library

- Η STL είναι μια βιβλιοθήκη επαναχρησιμοποιήσιμων στοιχείων που βασίζεται στο γενερικό προγραμματισμό (προγραμματισμό με templates).
- Η STL αποτελείται από
  - **Containers (περιέκτες)**: Επιτρέπουν την οργάνωση μιας συλλογής αντικειμένων στη μνήμη του Η/Υ. Πρόκειται για templated κλάσεις (π.χ. vector<int>, list<double>, …).
  - **Algorithms (αλγόριθμοι)**: Αλγόριθμοι που εφαρμόζονται σε containers (π.χ. sort, find, …). Είναι γενικοί και μπορούν να χρησιμοποιηθούν σε διάφορους τύπους containers.
  - **Iterators (επαναλήπτες)**: «Δείκτες» που επιτρέπουν τη διάσχιση και σε ορισμένες περιπτώσεις την αλλαγή ενός container. Ένας iterator δείχνει σε κάποιο στοιχείο ενός container.

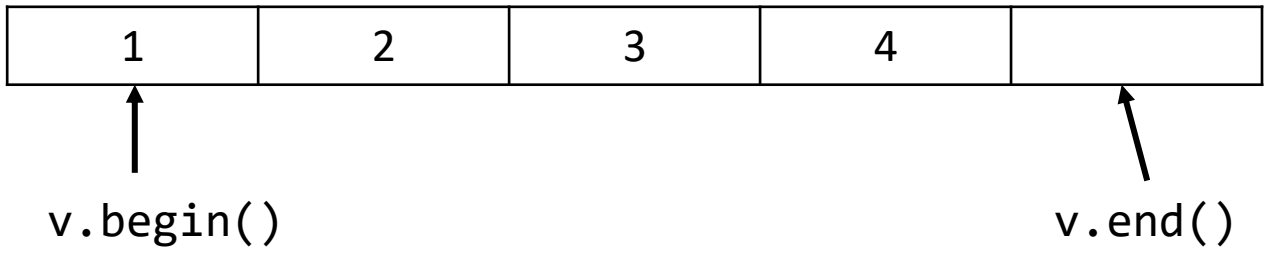
Containers

- Σε ένα container μπορούν να αποθηκευτούν τιμές βασικών τύπων καθώς και αντικείμενα.
- Τα containers χωρίζονται σε δύο βασικές κατηγορίες:
  - Ακολουθιακά containers (sequence containers) – κάθε αντικείμενο ακολουθείται από κάποιο άλλο αντικείμενο, έχουν δηλαδή γραμμική διευθέτηση.
    - vector (διάνυσμα)
    - array (διάνυσμα σταθερού μεγέθους)
    - list (διπλά συνδεδεμένη λίστα)
    - forward_list (απλά συνδεδεμένη λίστα)
    - deque (ουρά με δύο άκρα)
    - container adaptors
      - stack (στοίβα)
      - queue (ουρά)
      - priority_queue (ουρά προτεραιότητας)
  - Containers αντιστοίχισης (associative containers) – γίνεται χρήση κλειδιών για την πρόσβαση στα στοιχεία του container, επιτρέπουν τη γρήγορη πρόσβαση βάσει κλειδιών.
    - set (σύνολο)
    - multiset
    - map (πίνακας αντιστοίχισης)
    - multimap

std::vector

- Διάνυσμα με δυνατότητα δυναμικής αύξησης του μεγέθους του έτσι ώστε να δεχθεί νέα στοιχεία.
- Constructors
  - `vector<T> v; // άδειο vector`
  - `vector<T> v(n); // vector με n αντίγραφα της προκαθορισμένης τιμής του T`
  - `vector<T> v(n, value); // vector με n αντίγραφα της τιμής value`
- Προσθήκη νέων στοιχείων
  - `push_back()`
- Μέγεθος – χωρητικότητα
  - `size()`
  - `capacity()`

```
std::vector<int> v1{1, 2, 3, 4};
```



Δεικτοδότηση σε vector

- Δεικτοδότηση με []
  - Υψηλή απόδοση, η πρόσβαση γίνεται χωρίς έλεγχο ορίων του vector.
- Δεικτοδότηση με at()
  - Πραγματοποιεί έλεγχο ορίων, προκαλεί την εξαίρεση out_of_range exception αν επιχειρηθεί πρόσβαση εκτός των ορίων του vector.

https://chgogos.github.io/oop/cpp_playground/ex046/stl_vector1.cpp

https://chgogos.github.io/oop/cpp_playground/ex046/stl_vector2.cpp

https://chgogos.github.io/oop/cpp_playground/ex046/stl_vector3.cpp

https://chgogos.github.io/oop/cpp_playground/ex046/stl_vector4.cpp

https://chgogos.github.io/oop/cpp_playground/ex046/stl_vector5.cpp

std::array

- Διάνυσμα με προκαθορισμένο μέγεθος
- Βασικές λειτουργίες:
  - size()
  - τελεστής []

```
std::array<int, 5> a{1, 2, 3, 4, 5};
```

https://chgogos.github.io/oop/cpp_playground/ex046/stl_array.cpp

std::deque

- Διάνυσμα με δύο άκρα.
- Βασικές συναρτήσεις μέλη:
  - operator[]
  - at()
  - front()
  - push_front()
  - back()
  - push_back()
  - …

https://chgogos.github.io/oop/cpp_playground/ex046/stl_deque.cpp

std::forward_list

- Απλά συνδεδεμένη λίστα
- Βασικές συναρτήσεις μέλη:
  - front()
    - πρόσβαση στο πρώτο στοιχείο της λίστας
  - push_front()
    - εισαγωγή στοιχείου στην αρχή της λίστας
  - pop_front()
    - διαγραφή του πρώτου στοιχείου της λίστας
  - sort()
    - ταξινόμηση
  - remove_if()
    - διαγραφή στοιχείων που ικανοποιούν συγκεκριμένα κριτήρια
  - …

https://chgogos.github.io/oop/cpp_playground/ex046/stl_forward_list.cpp

std::list

- Διπλά συνδεδεμένη λίστα
- Βασικές συναρτήσεις μέλη:
  - front()
    - πρόσβαση στο πρώτο στοιχείο της λίστας
  - push_front()
    - εισαγωγή στοιχείου στην αρχή της λίστας
  - back()
    - πρόσβαση στο πρώτο στοιχείο της λίστας
  - push_back()
    - εισαγωγή στοιχείου στο τέλος της λίστας
  - sort()
    - ταξινόμηση
  - reverse()
    - Αντιστροφή
  - …

https://chgogos.github.io/oop/cpp_playground/ex046/stl_list.cpp

std::set

- Σε ένα set αποθηκεύονται αντικείμενα που καθένα από αυτά αποτελεί το ίδιο ή περιέχει ένα κλειδί.
  - Κάθε κλειδί μπορεί να υπάρχει μόνο μια φορά το πολύ.
  - Τα κλειδιά είναι ταξινομημένα.
- Το std::set συνήθως υλοποιείται ως ισορροπημένο δυαδικό δένδρο αναζήτησης.
- Παραλλαγές του set
  - multiset: επιτρέπει πολλές εμφανίσεις του ίδιου κλειδιού.
  - unordered_set: δεν υπάρχει διάταξη με βάση τα κλειδιά, υλοποιείται ως πίνακας κατακερματισμού.

https://chgogos.github.io/oop/cpp_playground/ex046/stl_set.cpp

https://chgogos.github.io/oop/cpp_playground/ex046/stl_set2.cpp

std::map

- Σε ένα map αποθηκεύονται ζεύγη (key, value), δηλαδή για κάθε κλειδί υπάρχει μια σχετιζόμενη τιμή.
  - Κάθε κλειδί μπορεί να υπάρχει μόνο μια φορά το πολύ.
  - Τα ζεύγη είναι ταξινομημένα με βάση τα κλειδιά.
- Παραλλαγές του map:
  - multimap: επιτρέπει πολλές εμφανίσεις του ίδιου κλειδιού.
  - unordered_map: δεν υπάρχει διάταξη με βάση τα κλειδιά, υλοποιείται ως πίνακας κατακερματισμού.

https://chgogos.github.io/oop/cpp_playground/ex046/stl_map.cpp

https://chgogos.github.io/oop/cpp_playground/ex046/stl_map2.cpp

https://chgogos.github.io/oop/cpp_playground/ex046/stl_multimap.cpp

Container Adaptor κλάσεις

- stack (στοίβα)
- queue (ουρά)
- priority_queue (ουρά προτεραιότητας)

https://chgogos.github.io/oop/cpp_playground/ex046/stl_stack.cpp

https://chgogos.github.io/oop/cpp_playground/ex046/stl_queue.cpp

https://chgogos.github.io/oop/cpp_playground/ex046/stl_priority_queue.cpp

Iterators

- Οι iterators είναι αντικείμενα που μοιάζουν με δείκτες και χρησιμοποιούνται για την πρόσβαση στα περιεχόμενα ενός container.
- Επιτρέπουν τη «διάσχιση» ενός container από το ένα στοιχείο του στο επόμενο.
- Οι συναρτήσεις `begin()` και `end()` επιτρέπουν την αναφορά στο πρώτο στοιχείο ενός container και στη θέση αμέσως μετά από το τελευταίο στοιχείο ενός container, αντίστοιχα. Εναλλακτικά μπορούν να χρησιμοποιηθούν οι συναρτήσεις μέλη `begin()` και `end()` για containers όπως το vector.

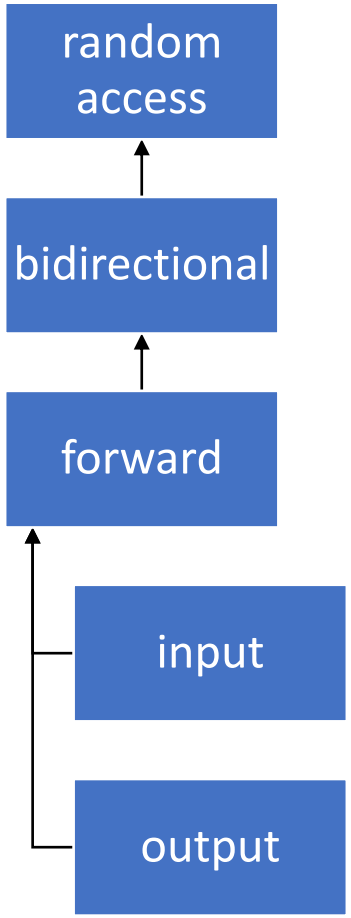
```
vector<int> v;
…
vector<int>::iterator iter=begin(v);
ή
auto iter = begin(v);
ή
auto iter = v.begin(); // συνάρτηση μέλος του vector
```

Iterators

- Οι iterators αποτελούν ένα μηχανισμό για τον καθορισμό μιας θέσης μέσα σε ένα container.
- Υπάρχουν διάφοροι τύποι iterators, αλλά όλοι χρησιμοποιούν τον ίδιο βασικό τρόπο διάσχισης και πρόσβασης στα στοιχεία ενός container, δηλαδή:
  - Τον τελεστή ++ για τη μετακίνηση στο επόμενο στοιχείο.
  - Τον τελεστή * για την πρόσβαση στο τρέχον στοιχείο.
  - Σύγκριση ενός iterator με έναν άλλο iterator.

Τύποι iterators

- Υπάρχουν συγκεκριμένοι iterators που μπορούν να χρησιμοποιηθούν με κάθε container.
- Υπάρχουν 5 βασικοί τύποι iterators: random access, bidirectional, forward, input, output



Τύπος iterator ανά container

Container	Iterator
vector	random access
deque	random access
list	bidirectional
set	bidirectional
multiset	bidirectional
map	bidirectional
multimap	bidirectional
forward_list	forward
unordered_set	forward

- Ο `vector<int>::iterator` είναι random access iterator για πρόσβαση σε ακεραίους.
- Ο `list<float>::iterator` είναι bidirectional iterator για πρόσβαση σε πραγματικούς.
- Ο `forward_list<student>::iterator` είναι forward iterator για πρόσβαση σε αντικείμενα student.
- Οι container adaptors (`stack, queue, priority_queue`) δεν έχουν iterators.

https://chgogos.github.io/oop/cpp_playground/ex046/stl_random_iterator.cpp

https://chgogos.github.io/oop/cpp_playground/ex046/stl_bidirectional_iterator.cpp

https://chgogos.github.io/oop/cpp_playground/ex046/stl_forward_iterator.cpp

Iterator adaptors για εισαγωγή (inserters – insert iterators)

- Οι insert iterators είναι adaptors που επιτρέπουν σε αλγορίθμους να λειτουργούν εισάγοντας επιπλέον στοιχεία στα containers αντί να ενημερώνουν τα ήδη υπάρχοντα στοιχεία.
- Έτσι οι αλγόριθμοι μπορούν να προσθέτουν στοιχεία σε container προορισμούς που δεν είναι ήδη επαρκώς μεγάλα για να δεχθούν τις νέες τιμές.

- Υπάρχουν 3 είδη inserters, ανάλογα με το container που θα δεχθεί τις τιμές:
  - back_inserter(), αν το container υποστηρίζει push_back()
  - front_inserter(), αν το container υποστηρίζει push_front()
  - inserter(), αν το container υποστηρίζει insert()

https://chgogos.github.io/oop/cpp_playground/ex046/insert_iterators.cpp

stream iterator

- ```
  vector<int> v{1,2,3,4,5};
  copy(v.begin(), v.end(), ostream_iterator<int>(cout, " "));
  ```

https://github.com/chgogos/oop/blob/master/cpp_playground/ex046/stream_iterator1.cpp

- ```
  vector<int> v;
  copy(istream_iterator<int>(cin), istream_iterator<int>(),
  back_inserter(v))
  ```

https://github.com/chgogos/oop/blob/master/cpp_playground/ex046/stream_iterator2.cpp

Χρήσιμες συναρτήσεις για iterators

- Μόνο οι random access iterators επιτρέπουν την πρόσθεση ή την αφαίρεση μιας ακέραιας τιμής καθώς και την αφαίρεση ενός iterator από έναν άλλο. Ωστόσο, η ίδια λειτουργικότητα μπορεί να επιτευχθεί σε άλλους iterators με τις συναρτήσεις:
  - `advance(iter, n)` που μετακινεί τον iterator προς τα μπροστά ή προς τα πίσω ανάλογα με το αν η παράμετρος n λάβει θετική ή αρνητική τιμή.
  - `distance(iter1, iter2)` που επιστρέφει το πλήθος θέσεων ανάμεσα στους δύο iterators (η τιμή που επιστρέφεται πρέπει να γίνει cast σε `int`).

https://chgogos.github.io/oop/cpp_playground/ex046/advance.cpp

https://chgogos.github.io/oop/cpp_playground/ex046/distance.cpp

Δείκτες συναρτήσεων (function pointers)

- Οι δείκτες συναρτήσεων είναι μεταβλητές που κρατούν τη διεύθυνση μιας συνάρτησης.
- «Περίεργη» σύνταξη
  - `int (*fp)(int, int);` // δήλωση function pointer με όνομα `fp` προς μια συνάρτηση που δέχεται 2 ακεραίους και επιστρέφει έναν ακέραιο.
  - Αν υπάρχει μια συνάρτηση της μορφής `int fun(int, int)` τότε μπορεί ο function pointer να δείχνει σε αυτή τη συνάρτηση:
    - `fp = fun;`
  - Η δε κλήση της συνάρτησης μέσω του function pointer γίνεται ως εξής:
    - `x = fp(2,3);` ή ως `x = (*fp)(2,3);`

https://chgogos.github.io/oop/cpp_playground/ex001/function_pointers1.cpp

https://chgogos.github.io/oop/cpp_playground/ex001/function_pointers2.cpp

https://chgogos.github.io/oop/cpp_playground/ex001/function_pointers3.cpp

Δείκτες συναρτήσεων (function pointers)

- Στη C++ 11 υπάρχει η δυνατότητα δήλωσης ενός function pointer με το `std::function` που ορίζεται στο header `<functional>`.
  - `std::function<int(int,int)> fp;` // δήλωση function pointer με όνομα `fp` προς μια συνάρτηση που δέχεται 2 ακεραίους και επιστρέφει έναν ακέραιο.

https://chgogos.github.io/oop/cpp_playground/ex001/function_pointers4.cpp

https://chgogos.github.io/oop/cpp_playground/ex001/function_pointers5.cpp

https://chgogos.github.io/oop/cpp_playground/ex001/function_pointers6.cpp

Αντικείμενα συναρτήσεων (functors = function objects)

- Functor είναι οποιοδήποτε αντικείμενο στο οποίο έχει υπερφορτωθεί ο τελεστής ( ) και συνεπώς μπορεί να χρησιμοποιηθεί με παρενθέσεις σαν να είναι συνάρτηση.
- Ένα functor μπορεί να διατηρεί κατάσταση καθώς ως αντικείμενο μπορεί να διαθέτει μέλη δεδομένων.
- Συνήθως ένα functor είναι ταχύτερο από ένα function pointer.

Functors που ορίζονται από το χρήστη

```
class FunctionObjectType
{
public:
     τύπος_επιστροφής operator()(παράμετροι)
     {
     }
};
```

https://chgogos.github.io/oop/cpp_playground/ex046/functor1.cpp

https://chgogos.github.io/oop/cpp_playground/ex046/functor2.cpp

https://chgogos.github.io/oop/cpp_playground/ex046/functor3.cpp

Τύποι functors – η περίπτωση των predicates

- Στην STL υπάρχουν τα ακόλουθα functors:
  - generators, πρόκειται για functors που καλούνται χωρίς παραμέτρους.
  - unary functions, πρόκειται για functors που καλούνται με μια παράμετρο.
  - binary functions , πρόκειται για functors που καλούνται με δύο παραμέτρους.
- Τα κατηγορήματα (predicates) είναι functors που επιστρέφουν μια λογική τιμή.
  - Μια unary function που επιστρέφει μια λογική τιμή είναι ένα predicate.
  - Μια binary function που επιστρέφει μια λογική τιμή είναι ένα binary predicate.
- Παράδειγμα χρήσης functor με τη remove_if

https://chgogos.github.io/oop/cpp_playground/ex046/remove_if_functor.cpp

Functors της STL (1/2)

- **Αριθμητικά δυαδικά κατηγορήματα**
  - **plus<T> f;**
    - **f(x,y) επιστρέφει x + y.**
  - **minus<T> f;**
    - **f(x,y) επιστρέφει x - y.**
  - **multiplies<T> f;**
    - **f(x,y) επιστρέφει x - y.**
  - **divides<T> f;**
    - **f(x,y) επιστρέφει x / y.**
  - **modulus<T> f;**
    - **f(x,y) επιστρέφει x % y.**
- **Σχεσιακά δυαδικά κατηγορήματα**
  - **equal_to<T> f;**
    - **f(x,y) επιστρέφει x == y.**
  - **not_equal_to<T> f;**
    - **f(x,y) επιστρέφει x != y.**
  - **less<T> f;**
    - **f(x,y) επιστρέφει x < y.**
  - **less_equal<T> f;**
    - **f(x,y) επιστρέφει x <= y.**
  - **greater<T> f;**
    - **f(x,y) επιστρέφει x > y.**
  - **greater_equal<T> f;**
    - **f(x,y) επιστρέφει x >= y.**

Functors της STL (2/2)

- **Λογικά δυαδικά κατηγορήματα**
  - **logical_and<T> f;**
    - **f(x,y) επιστρέφει x && y.**
  - **logical_or<T> f;**
    - **f(x,y) επιστρέφει x || y.**
- **Αριθμητικά μοναδιαία functors**
  - **negate<T> f;**
    - **f(x) επιστρέφει -x.**
  - **logical_or<T> f;**
    - **f(x) επιστρέφει !x.**

https://chgogos.github.io/oop/cpp_playground/ex046/functor4.cpp

https://chgogos.github.io/oop/cpp_playground/ex046/functor5.cpp

Λάμδας (lambdas)

- Λάμδα (lambda ή closure) είναι μια ανώνυμη συνάρτηση που μπορεί να γραφεί απευθείας ως παράμετρος ή γενικότερα να χρησιμοποιηθεί σε μια έκφραση.

```
auto my_lambda = [](int x)->int {return x*2;};
cout << my_lambda(5); // εμφανίζει την τιμή 10
```

Σύνταξη λάμδα συναρτήσεων: [](){}

- [captures](parameters)->return_type {statements;}
- **[captures]**
  - Ποιες μεταβλητές έξω από τη λάμδα θα είναι διαθέσιμες στις εντολές της λάμδα και αν αυτές οι μεταβλητές θα περνούν με τιμή ή με αναφορά.
- **(parameters)**
  - παράμετροι που θα χρειάζονται για την κλήση της λάμδα (μπορεί η λίστα παραμέτρων να είναι κενή).
- **-> ret**
  - προαιρετικό, αν παραληφθεί ο μεταγλωττιστής «διαπιστώνει» τον τύπο επιστροφής από τις εντολές return που περιέχονται στο σώμα της λάμδα.
- **{statements;}**
  - Το σώμα της λάμδα.

https://chgogos.github.io/oop/cpp_playground/ex071/lambda1.cpp

https://chgogos.github.io/oop/cpp_playground/ex071/lambda2.cpp

https://chgogos.github.io/oop/cpp_playground/ex071/lambda3.cpp

https://chgogos.github.io/oop/cpp_playground/ex071/lambda4.cpp

function pointers vs function objects (functors) vs lambdas

- Στο παράδειγμα fp_functor_lambda.cpp όλα τα στοιχεία ενός `std::vector<int>` διπλασιάζονται. Η εργασία γίνεται με τη χρήση της συνάρτηση `transform()` που δέχεται ως 4η παράμετρο έναν function pointer ή έναν functor ή ένα lambda.

https://chgogos.github.io/oop/cpp_playground/ex087/fp_functor_lambda.cpp

Αλγόριθμοι: sort(), merge()

- Η sort() ταξινομεί μια περιοχή τιμών.
- Η stable_sort() ταξινομεί διατηρώντας τη μεταξύ τους σειρά για στοιχεία που έχουν το ίδιο κλειδί.
- Η merge() συγχωνεύει δύο ταξινομημένες περιοχές τιμών σε μια.

https://github.com/chgogos/oop/blob/master/cpp_playground/ex086/stl_sort.cpp

https://github.com/chgogos/oop/blob/master/cpp_playground/ex086/stl_stable_sort.cpp

https://github.com/chgogos/oop/blob/master/cpp_playground/ex086/stl_merge.cpp

Αλγόριθμοι: find(), find_if(), find_if_not()

- Η find() επιστρέφει έναν iterator στο πρώτο στοιχείο μιας περιοχής τιμών που ισούται με την παράμετρο που δέχεται.
- Η find_if() επιστρέφει έναν iterator στο πρώτο στοιχείο μιας περιοχής τιμών για την οποία το predicate που δέχεται ως παράμετρο είναι αληθές.
- Η find_if_not() επιστρέφει έναν iterator στο πρώτο στοιχείο μιας περιοχής τιμών για την οποία το predicate που δέχεται ως παράμετρο είναι ψευδές.

https://github.com/chgogos/oop/blob/master/cpp_playground/ex086/stl_find.cpp

https://github.com/chgogos/oop/blob/master/cpp_playground/ex086/stl_set_find.cpp

Αλγόριθμοι: count(), count_if()

- Η count() καταμετρά τις εμφανίσεις μιας τιμής σε μια περιοχή τιμών.
- Η count_if() καταμετρά τα στοιχεία σε μια περιοχή τιμών για τα οποία το predicate που δέχεται ως παράμετρο είναι αληθές.

https://github.com/chgogos/oop/blob/master/cpp_playground/ex086/stl_count.cpp

Αλγόριθμοι: for_each()

- Εφαρμόζει ένα lambda σε κάθε στοιχείο μιας περιοχής.

```
vector<int> v{4, 3, 6, 1, 2, 8, 7};
for_each(v.begin(), v.end(), [](int &x) { x++; });
```

https://github.com/chgogos/oop/blob/master/cpp_playground/ex086/stl_for_each.cpp

erase remove idiom

- Η διαγραφή ενός συνόλου τιμών από ένα vector με βάση κάποιο predicate γίνεται ως εξής:

```
vector<int> v{7,13,2,8,9,1,8,4,5};
// διαγραφή περιττών τιμών από το v
v.erase(remove_if(v.begin(), v.end(), [](int x){return x%2==1;}), v.end());
```

```
2 8 8 4
```

map – filter – reduce

- Μια συχνά χρησιμοποιούμενη αλληλουχία ενεργειών είναι η map – filter – reduce:
  - **map**: αντιστοίχιση κάθε τιμής σε κάποια άλλη τιμή.
  - **filter**: φιλτράρισμα τιμών έτσι ώστε να διατηρηθεί ένα υποσύνολο από αυτές.
  - **reduce**: συνάθροιση όλων των τιμών σε μια.
- Η αλληλουχία ενεργειών map – filter – reduce στη C++ υλοποιείται ως εξής:
  - map: `transform()`
  - filter: `remove_if() + erase()`
  - reduce: `accumulate()`

https://chgogos.github.io/oop/cpp_playground/ex071/map_filter_reduce.cpp

Άλλοι αλγόριθμοι της STL

- Υπάρχει πληθώρα επιπλέον αλγορίθμων στην STL (> 100)

Αναφορές

- http://cs.stmarys.ca/~porter/csc/ref/stl/
- https://www.learncpp.com/
- https://www.eventhelix.com/RealtimeMantra/Object_Oriented/stl-tutorial.htm
- http://www.math.univ-toulouse.fr/~grundman/stl-tutorial/tutorial.html