

# STL

# Standard Template Library

Τμήμα Πληροφορικής και Τηλεπικοινωνιών (Άρτα)

Πανεπιστήμιο Ιωαννίνων

Γκόγκος Χρήστος

# 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.

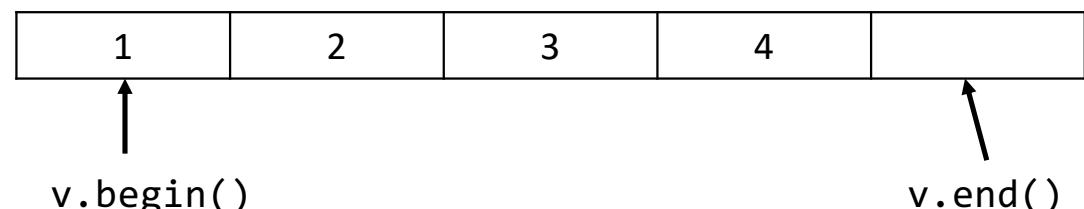
# Containers

- 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; // áδειο 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_vector1.cpp)

[https://chgogos.github.io/oop/cpp\\_playground/ex046/stl\\_vector2.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_vector3.cpp)

[https://chgogos.github.io/oop/cpp\\_playground/ex046/stl\\_vector4.cpp](https://chgogos.github.io/oop/cpp_playground/ex046/stl_vector4.cpp)

[https://chgogos.github.io/oop/cpp\\_playground/ex046/stl\\_vector5.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](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](https://chgogos.github.io/oop/cpp_playground/ex046/stl_deque.cpp)

# Άσκηση #1: std::deque

- Να γράψετε ένα πρόγραμμα που να διαχειρίζεται μετρήσεις θερμοκρασίας σε πραγματικό χρόνο χρησιμοποιώντας έναν container τύπου std::deque<double>. Το πρόγραμμα να δέχεται συνεχώς από τον χρήστη τιμές θερμοκρασίας, τις οποίες θα προσθέτει στο τέλος του deque, μέχρι ο χρήστης να εισάγει Ctrl+D, οπότε η εισαγωγή τερματίζει. Με κάθε νέα μέτρηση, εάν το πλήθος των στοιχείων ξεπερνά τα 5, θα αφαιρείται αυτόματα η παλαιότερη μέτρηση (από την αρχή του deque). Μετά από κάθε εισαγωγή, θα εμφανίζεται το τρέχον περιεχόμενο του deque, ενώ στο τέλος του προγράμματος θα υπολογίζεται και θα εμφανίζεται η μέση τιμή των τελευταίων 5 (ή λιγότερων) μετρήσεων.

# Άσκηση #1: Λύση

```
#include <deque>
#include <iostream>
#include <limits>
int main() {
    std::deque<double> temps;
    double value;
    while (true) {
        std::cout << "Enter temperature (negative to stop): ";
        if (!(std::cin >> value)) {
            if (std::cin.eof()) {
                std::cout << "\nEnd of input.\n"; // Ctrl+D για Linux,
                                                // Ctrl+Z και Enter για Windows
            }
            std::cin.clear(); // για μη έγκυρη είσοδο
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cout << "Invalid input, please enter a number.\n";
            continue;
        }
        temps.push_back(value);
        if (temps.size() > 5) temps.pop_front();
        std::cout << "Current deque: ";
        for (double t : temps) std::cout << t << " ";
        std::cout << std::endl;
    }
    if (!temps.empty()) {
        double sum = 0.0;
        for (double t : temps) sum += t;
        double avg = sum / temps.size();
        std::cout << "Average of last " << temps.size() << " measurements: " << avg
              << std::endl;
    } else {
        std::cout << "No measurements entered." << std::endl;
    }
    return 0;
}
```

```
Enter temperature (Ctrl+D stops): -1
Current deque: 4 -1
Enter temperature (Ctrl+D stops): 3
Current deque: 4 -1 3
Enter temperature (Ctrl+D stops): 2
Current deque: 4 -1 3 2
Enter temperature (Ctrl+D stops): 0
Current deque: 4 -1 3 2 0
Enter temperature (Ctrl+D stops): 5
Current deque: -1 3 2 0 5
Enter temperature (Ctrl+D stops): ^D
End of input.
Average of last 5 measurements: 1.8
```

# 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://chogogos.github.io/oop/cpp\\_playground/ex046/stl\\_forward\\_list.cpp](https://chogogos.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](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 (Balanced Binary Search Tree).
- 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_set.cpp)

[https://chgogos.github.io/oop/cpp\\_playground/ex046/stl\\_set2.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_map.cpp)

[https://chgogos.github.io/oop/cpp\\_playground/ex046/stl\\_map2.cpp](https://chgogos.github.io/oop/cpp_playground/ex046/stl_map2.cpp)

[https://chgogos.github.io/oop/cpp\\_playground/ex046/stl\\_multimap.cpp](https://chgogos.github.io/oop/cpp_playground/ex046/stl_multimap.cpp)

## Άσκηση #2: std::map

- Να γράψετε ένα πρόγραμμα που να υπολογίζει το πλήθος εμφανίσεων των πεζών γραμμάτων του αγγλικού αλφαβήτου σε ένα κείμενο χρησιμοποιώντας έναν container τύπου std::map<char, int>.
- Το πρόγραμμα να δέχεται από τον χρήστη ένα αρχείο κειμένου ως είσοδο με ανακατεύθυνση εισόδου (redirected input) με το <.
- Για κάθε χαρακτήρα του κειμένου που περιέχει το αρχείο εισόδου, το πρόγραμμα να ενημερώνει το std::map αυξάνοντας τον μετρητή εμφανίσεων του κατάλληλου γράμματος κατά 1.
- Το πρόγραμμα να εμφανίζει όλα τα γράμματα αλφαριθμητικά μαζί με το πλήθος των εμφανίσεων τους.
- Το πρόγραμμα θα πρέπει να ελέγχει την εγκυρότητα της εισόδου και αν εκτελεστεί χωρις είσοδο θα εμφανίζει μήνυμα σφάλματος.
- Το πρόγραμμα θα εκτελείται ως εξής:  
\$ ./a.out < test.txt

# Άσκηση #2: Λύση

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

int main() {
    map<char, int> freq;
    string token;
    if (!(cin >> token)) {
        cerr << "Error: no input provided. Use: ./a.out < filename\n";
        return 1;
    }
    for (char ch : token) {
        if (ch >= 'a' && ch <= 'z') freq[ch]++;
    }
    while (cin >> token) {
        for (char ch : token) {
            if (ch >= 'a' && ch <= 'z') freq[ch]++;
        }
    }
    cout << "Letter frequencies:\n";
    for (char c = 'a'; c <= 'z'; ++c) {
        cout << c << ":" << freq[c] << "\n";
    }
    return 0;
}
```

```
$ g++ oop16_ask2.cpp < oop16_ask2.cpp
Letter frequencies:
a: 15
b: 0
c: 27
d: 6
e: 28
f: 12
g: 3
h: 13
i: 20
j: 0
k: 5
l: 5
m: 6
n: 26
o: 15
p: 5
q: 5
r: 26
s: 8
t: 19
u: 11
v: 1
w: 1
x: 0
y: 0
z: 3
```

# 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_stack.cpp)

[https://chgogos.github.io/oop/cpp\\_playground/ex046/stl\\_queue.cpp](https://chgogos.github.io/oop/cpp_playground/ex046/stl_queue.cpp)

[https://chgogos.github.io/oop/cpp\\_playground/ex046/stl\\_priority\\_queue.cpp](https://chgogos.github.io/oop/cpp_playground/ex046/stl_priority_queue.cpp)

## Άσκηση #3: std::priority\_queue

- Να γράψετε ένα πρόγραμμα που να προσομοιώνει το σύστημα διαλογής ασθενών σε ένα τμήμα επειγόντων περιστατικών νοσοκομείου χρησιμοποιώντας έναν container τύπου std::priority\_queue.
- Κάθε ασθενής που φτάνει χαρακτηρίζεται από το όνομά του και έναν ακέραιο βαθμό σοβαρότητας από 0 μέχρι και 100, όπου μεγαλύτερη τιμή σημαίνει πιο επείγον περιστατικό.
- Το πρόγραμμα θα εμφανίζει μενού με τις ακόλουθες επιλογές:
  - 1. ARRIVE όνομα σοβαρότητα για εισαγωγή νέου ασθενούς στην ουρά προτεραιότητας,
  - 2. TREAT για εμφάνιση και αφαίρεση του ασθενούς με τη μεγαλύτερη σοβαρότητα,
  - 3. STATUS για εμφάνιση του πλήθους των ασθενών που περιμένουν, και
  - 4. EXIT για τερματισμό του προγράμματος.

# Άσκηση #3: Λύση

```
#include <iostream>
#include <queue>
#include <string>

using namespace std;

struct Patient {
    string name;
    int severity; // 0-100
};

struct ComparePatients {
    bool operator()(const Patient& a, const Patient& b) const {
        return a.severity < b.severity;
    }
};

int main() {
    priority_queue<Patient, vector<Patient>, ComparePatients> pq;
    int choice;
    while (true) {
        cout << "\n--- Emergency Department Triage System ---\n"
            << "1. ARRIVE\n"
            << "2. TREAT\n"
            << "3. STATUS\n"
            << "4. EXIT\n"
            << "Select an option: ";
        cin >> choice;
        if (!cin) {
            cerr << "Invalid input. Exiting.\n";
            return 1;
        }
    }
}
```

```
if (choice == 1) {
    string name;
    int severity;
    cout << "Enter patient name: ";
    cin >> name;
    cout << "Enter severity (0-100): ";
    cin >> severity;
    if (!cin || severity < 0 || severity > 100) {
        cout << "Invalid severity value.\n";
        continue;
    }
    pq.push(Patient{name, severity});
    cout << "Patient added to queue.\n";
} else if (choice == 2) {
    if (pq.empty()) {
        cout << "No patients waiting.\n";
    } else {
        Patient p = pq.top();
        pq.pop();
        cout << "Treating patient: " << p.name << " with severity "
            << p.severity << "\n";
    }
} else if (choice == 3) {
    cout << "Number of patients waiting: " << pq.size() << "\n";
} else if (choice == 4) {
    cout << "Exiting program.\n";
    break;
} else {
    cout << "Invalid choice.\n";
}
}

return 0;
}
```

# 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`.

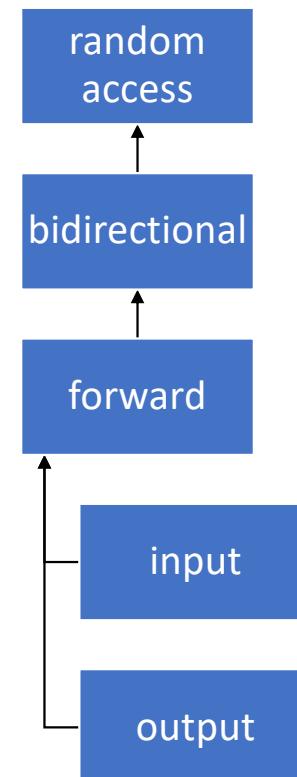
```
vector<int> v;  
...  
vector<int>::iterator iter=begin(v);  
ή  
auto iter = begin(v);  
ή  
auto iter = v.begin(); // συνάρτηση μέλος του vector
```

# 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://chgomos.github.io/oop/cpp\\_playground/ex046/stl\\_random\\_iterator.cpp](https://chgomos.github.io/oop/cpp_playground/ex046/stl_random_iterator.cpp)

[https://chgomos.github.io/oop/cpp\\_playground/ex046/stl\\_bidirectional\\_iterator.cpp](https://chgomos.github.io/oop/cpp_playground/ex046/stl_bidirectional_iterator.cpp)

[https://chgomos.github.io/oop/cpp\\_playground/ex046/stl\\_forward\\_iterator.cpp](https://chgomos.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](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](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](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(iterator, 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(iterator1, iterator2)` που επιστρέφει το πλήθος θέσεων ανάμεσα στους δύο iterators (η τιμή που επιστρέφεται πρέπει να γίνει cast σε `int`).

[https://chogogos.github.io/oop/cpp\\_playground/ex046/advance.cpp](https://chogogos.github.io/oop/cpp_playground/ex046/advance.cpp)

[https://chogogos.github.io/oop/cpp\\_playground/ex046/distance.cpp](https://chogogos.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_pointers1.cpp)  
[https://chgogos.github.io/oop/cpp\\_playground/ex001/function\\_pointers2.cpp](https://chgogos.github.io/oop/cpp_playground/ex001/function_pointers2.cpp)  
[https://chgogos.github.io/oop/cpp\\_playground/ex001/function\\_pointers3.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_pointers4.cpp)

[https://chgogos.github.io/oop/cpp\\_playground/ex001/function\\_pointers5.cpp](https://chgogos.github.io/oop/cpp_playground/ex001/function_pointers5.cpp)

[https://chgogos.github.io/oop/cpp\\_playground/ex001/function\\_pointers6.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/funcor1.cpp](https://chgogos.github.io/oop/cpp_playground/ex046/funcor1.cpp)

[https://chgogos.github.io/oop/cpp\\_playground/ex046/funcor2.cpp](https://chgogos.github.io/oop/cpp_playground/ex046/funcor2.cpp)

[https://chgogos.github.io/oop/cpp\\_playground/ex046/funcor3.cpp](https://chgogos.github.io/oop/cpp_playground/ex046/funcor3.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://chogos.github.io/oop/cpp\\_playground/ex046/remove\\_if\\_functor.cpp](https://chogos.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  $x \leq y$ .
  - `greater<T> f;`
    - $f(x,y)$  επιστρέφει  $x > y$ .
  - `greater_equal<T> f;`
    - $f(x,y)$  επιστρέφει  $x \geq y$ .

# 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  $x \mid\mid y$ .
- Αριθμητικά μοναδιαία functors
  - `negate<T> f;`
    - $f(x)$  επιστρέφει  $-x$ .
  - `logical_not<T> f;`
    - $f(x)$  επιστρέφει  $\neg x$ .

[https://chgogos.github.io/oop/cpp\\_playground/ex046/func4.cpp](https://chgogos.github.io/oop/cpp_playground/ex046/func4.cpp)

[https://chgogos.github.io/oop/cpp\\_playground/ex046/func5.cpp](https://chgogos.github.io/oop/cpp_playground/ex046/func5.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){return x*2;};  
cout << my_lambda(5); // εμφανίζει την τιμή 10
```

# Σύνταξη λάμδα συναρτήσεων: [](){} [captures](parameters)->return\_type {statements;}

- [captures](parameters)->return\_type  
{statements;}
- **[captures]**
  - Ποιες μεταβλητές έχω από τη λάμδα θα είναι διαθέσιμες στο σώμα 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).
- **-> return\_type**
  - προαιρετικό, αν δεν υπάρχει, ο μεταγλωττιστής «διαπιστώνει» τον τύπο επιστροφής από τις εντολές return που περιέχονται στο σώμα της λάμδα.
- **{statements;}**
  - Το σώμα της λάμδα.

[https://chgomgos.github.io/oop/cpp\\_playground/ex071/lambda1.cpp](https://chgomgos.github.io/oop/cpp_playground/ex071/lambda1.cpp)

[https://chgomgos.github.io/oop/cpp\\_playground/ex071/lambda2.cpp](https://chgomgos.github.io/oop/cpp_playground/ex071/lambda2.cpp)

[https://chgomgos.github.io/oop/cpp\\_playground/ex071/lambda3.cpp](https://chgomgos.github.io/oop/cpp_playground/ex071/lambda3.cpp)

[https://chgomgos.github.io/oop/cpp\\_playground/ex071/lambda4.cpp](https://chgomgos.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 4<sup>η</sup> παράμετρο έναν function pointer ή έναν functor ή ένα lambda.

[https://chgogos.github.io/oop/cpp\\_playground/ex087/fp\\_functor\\_lambda.cpp](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_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_stable_sort.cpp)

[https://github.com/chgogos/oop/blob/master/cpp\\_playground/ex086/stl\\_merge.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_find.cpp)

[https://github.com/chgogos/oop/blob/master/cpp\\_playground/ex086/stl\\_set\\_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](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](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());
```

2884

# 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](https://chgogos.github.io/oop/cpp_playground/ex071/map_filter_reduce.cpp)

## Άλλοι αλγόριθμοι της STL

- Υπάρχει πληθώρα επιπλέον αλγορίθμων στην STL (> 100)

<https://en.cppreference.com/w/cpp/algorithm.html>

# Αναφορές

- <https://www.fluentcpp.com/stl/>
- <https://www.learnCPP.com/>
- [https://www.eventhelix.com/RealtimeMantra/Object Oriented/stl-tutorial.htm](https://www.eventhelix.com/RealtimeMantra/Object_Oriented/stl-tutorial.htm)
- [TOPCODER - POWER UP C++ WITH THE STANDARD TEMPLATE LIBRARY PART ONE](#)
- [TOPCODER - POWER UP C++ WITH THE STANDARD TEMPLATE LIBRARY PART TWO](#)