ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΤΕΧΝΙΚΕΣ 2024-2025

https://helios.ntua.gr/course/view.php?id=869

Διδάσκοντες:

Βασίλης Βεσκούκης, Καθ.
Νίκος Λεονάρδος, Επ.Καθ.
Άρης Παγουρτζής, Καθ.
Νίκος Παπασπύρου, Καθ.
Σωτήρης Κοκόσης, ΕΔΙΠ
Πέτρος Ποτίκας, ΕΔΙΠ
Γιώργος Σιόλας, ΕΔΙΠ

progtech@courses.softlab.ntua.gr

21/3/2025

Διαφάνειες παρουσιάσεων

- ✓ Η γλώσσα προγραμματισμού C++
- ✓ Αρχές αντικειμενοστρεφούς προγραμματισμού
- ✓ Δομές δεδομένων

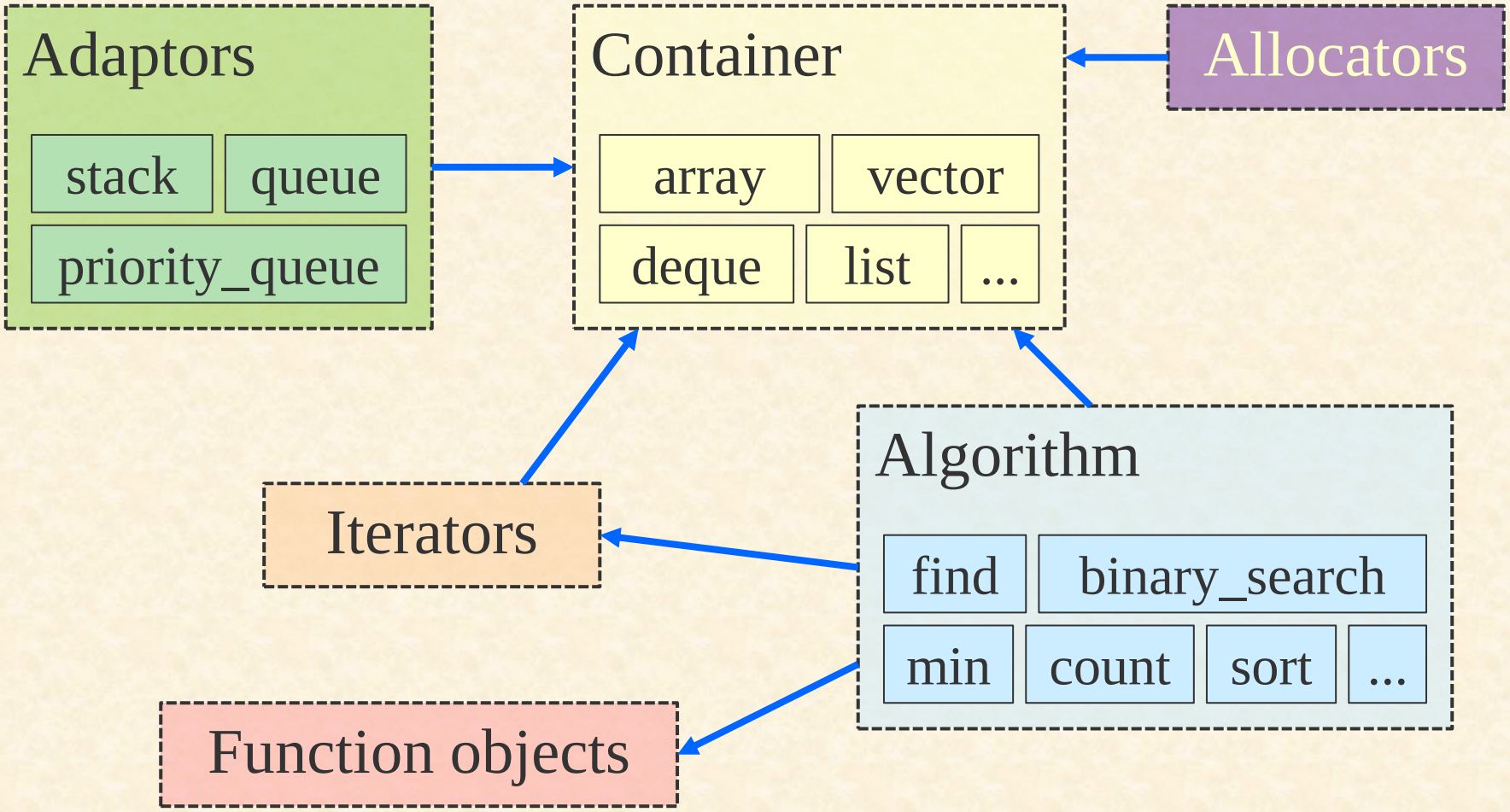
Standard Template Library

Εισαγωγή στην STL

Εισαγωγή στην STL

- Standard Template Library
- Μέρος της βιβλιοθήκης της C++
- Αρχική σχεδίαση: Alexander Stepanov
- Περιέχει χρήσιμες υλοποιήσεις:
  - δομών δεδομένων
  - αλγορίθμων
- Για να καταλάβουμε σε βάθος την STL:
  - templates
  - operator overloading
  - ιεραρχίες κλάσεων

Εισαγωγή στην STL



Εισαγωγή στην STL

www.cplusplus.com/reference/stl/

Container class templates

Sequence containers:

array C++11	Array class (class template)
vector	Vector (class template)
deque	Double ended queue (class template)
forward_list C++11	Forward list (class template)
list	List (class template)

Container adaptors:

stack	LIFO stack (class template)
queue	FIFO queue (class template)
priority_queue	Priority queue (class template)

Associative containers:

set	Set (class template)
multiset	Multiple-key set (class template)
map	Map (class template)
multimap	Multiple-key map (class template)

Unordered associative containers:

unordered_set C++11	Unordered Set (class template)
unordered_multiset C++11	Unordered Multiset (class template)
unordered_map C++11	Unordered Map (class template)
unordered_multimap C++11	Unordered Multimap (class template)

Συμβολοσειρά

- Τύπος **`string`**
  - περιτύλιγμα (wrapper) για array από **`char`**
- Χρήσιμες μέθοδοι
  - **`size`**, **`length`**
  - **`c_str`**
  - **`empty`**
  - **`operator[]`**, **`substr`**
  - **`append`**, **`operator+=`**
  - **`compare`**, **`operator==`**, **`operator<`**, κ.λπ.
  - **`insert`**, **`replace`**
  - **`find`**, **`rfind`**

Συμβολοσειρά

```
string s = "to be, or not to be? that is the question!";
string q = s;
cout << q << endl;
```

```
to be, or not to be? that is the question!
```

```
q += " _W.Shakespeare!";
cout << q << endl;
```

```
to be, or not to be? that is the question! _W.Shakespeare!
```

```
cout << "size=" << s.size()
     << " length=" << s.length() << endl;
```

```
size=42 length=42
```

```
cout << s[1] << endl;
```

```
o
```

```
cout << s.substr(3, 2) << endl;
```

```
be
```

```
s.replace(21, 4, "maybe THAT");
cout << s << endl;
```

```
to be, or not to be? maybe THAT is the question!
```

Συμβολοσειρά

```
s.insert(34, " indeed");
cout << s << endl;
```

```
to be, or not to be? maybe that is indeed the question!
```

```
int w = s.find("indeed", 0);
cout << w << endl;
```

```
35
```

```
w = s.find("really", 0);
cout << w << endl;
```

```
-1
```

```
const char *raw_pointer = s.data();  // or s.c_str()
cout << raw_pointer << endl;
```

```
to be, or not to be? maybe that is indeed the question!
```

```
w = s.find("o");
int x = s.rfind("o");
cout << w << " " << x << endl;
```

```
1 52
```

Containers: Πίνακας

- Τύπος **`array<T, n>`**
  - παρόμοια συμπεριφορά με τους απλούς πίνακες
  - σταθερού μεγέθους, γνωστού at compile time

```
#include <array>
...
array<int, 10> a;    // παρόμοιο με int a[10];
array<string, 20> s;
a[1] = 42;
s[2] = "Hello";
```

Containers: Πίνακας

- Χρήσιμες μέθοδοι της κλάσης array
  - size(): επιστρέφει το μέγεθος του πίνακα
  - at(): λειτουργεί ως getter και setter
  - operator[]: όπως το at() αλλά χωρίς έλεγχο ορίων

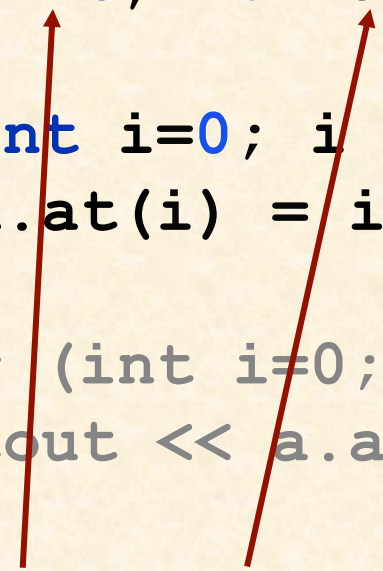
```
array<int, 10> a;
for (int i=0; i < a.size(); i++)
     a.at(i) = i * 10;                    // set
for (int i=0; i < a.size(); i++)
     cout << a.at(i) << " ";              // get
```

```
0 10 20 30 40 50 60 70 80 90
```

Containers: Πίνακας

- **for()** με iterator(*)

```
array<int, 10> a;

for (int i=0; i < a.size(); i++)
      a.at(i) = i * 10;

// for (int i=0; i < a.size(); i++)
//    cout << a.at(i) << " ";

for (int d : a)
      cout << d << " ";
```



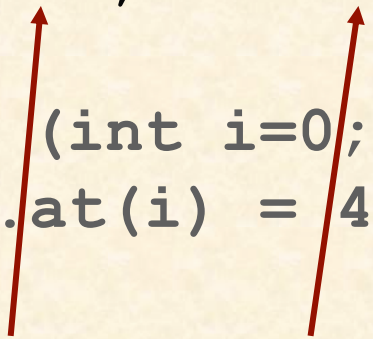
```
for (type var : container)
   ... do something
```

```
0 10 20 30 40 50 60 70 80 90
```

Containers: Πίνακας

- **for()** με iterator(*)

```
array<int, 10> a;

// for (int i=0; i < a.size(); i++)
//    a.at(i) = 42;

for (int &d : a)
     d = 42;

for (int d : a)
     cout << d << " ";
```



```
for (type var : container)
   ... do something
```

```
0 10 20 30 40 50 60 70 80 90
```

Containers: Πίνακας

- 2+ διαστάσεις

```
array<int, 10> a;

array<array<int, 10>, 10> b;

for (int i=0; i<10; i++)
    for (int j=0; j<10; j++)
        b[i][j] = i + j;
     // b.at(i).at(j) = i + j;

for (array<int,10> row : b) {
    for (int d : row)
        cout << d << "\t";
    cout << endl;
}

for (const array<int,10> &row : b) ...
```

0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10
2	3	4	5	6	7	8	9	10	11
3	4	5	6	7	8	9	10	11	12
4	5	6	7	8	9	10	11	12	13
5	6	7	8	9	10	11	12	13	14
6	7	8	9	10	11	12	13	14	15
7	8	9	10	11	12	13	14	15	16
8	9	10	11	12	13	14	15	16	17
9	10	11	12	13	14	15	16	17	18

Containers: Διάνυσμα

- Τύπος **`vector<T>`**
  - παρόμοια συμπεριφορά με array
  - μεταβλητού μεγέθους
    (εισαγωγή και αφαίρεση στο τέλος*)
  - **`push_back`, `pop_back`**
  - ειδική αποδοτική υλοποίηση: **`vector<bool>`**

```
void showVector(const vector<int> &v);
```

```
vector<int> v;
for (int i=0; i<5; i++) v.push_back(i);
showVector(v);
v.push_back(5); showVector(v);
v.pop_back(); showVector(v);
```

```
size= 5  contents= 0 1 2 3 4
size= 6  contents= 0 1 2 3 4 5
size= 5  contents= 0 1 2 3 4
```

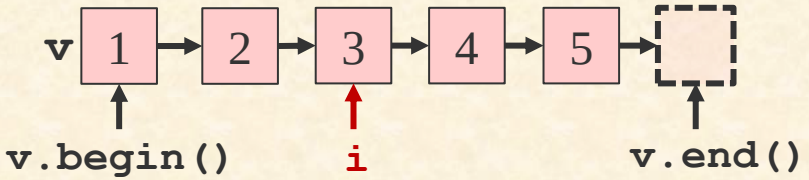
Containers: Διάνυσμα

```
#include <vector>
...
                    ΠΛΗΘΟΣ                ΑΡΧΙΚΗ ΤΙΜΗ
vector<int> x(100), y(200, 42);
cout << y[199] << endl;                  // 42
cout << x.size() << endl;                // 100

x.push_back(17);
cout << x[100] << endl;                  // 17
cout << x.size() << endl;                // 101

x.pop_back();
cout << x.size() << endl;                // 100
```

Iterators

- Iterators (πχ) **`vector<T>::iterator`**
  - επιτρέπουν τη διάσχιση ενός container iterable object
  - συμπεριφέρονται σαν να ήταν δείκτες

```
void showVector(const vector<int> &v) {
  cout << "size= " << v.size() << "  contents= ";
  vector<int>::iterator it;
  for (it = v.begin(); it != v.end(); ++it)
      cout << *it << " ";
  cout << endl;
}
```



Iterators

- Εναλλακτική, απλούστερη διάσχιση (C++11)

```
vector<int> v;

for (int x : v)
  cout << x << endl;

for (int &x : l)
  x = 42;

for (const int &x : l)
  cout << x << endl;


void showVector(const vector<int> &v) {
  cout << "size= " << v.size() << "  contents= ";
  for (int d : v) cout << d << " ";
  cout << endl;
}
```

Containers: Διάνυσμα

```
vector<int> v;
for (int i=0; i<5; i++) v.push_back(i);
showVector(v);
```
```
size= 5  contents= 0 1 2 3 4
```

```
cout << v.capacity() << endl;
```
```
8
```
γενικά ≥ size, implementation dependent

```
v.push_back(5); showVector(v);
```
```
size= 6  contents= 0 1 2 3 4 5
```

```
v.pop_back(); showVector(v);
```
```
size= 5  contents= 0 1 2 3 4
```

```
v.insert(v.begin()+2, 40);
showVector(v);
```
```
size= 6  contents= 0 1 40 2 3 4
```

```
vector<int> r(5, 10);
showVector(r);
```
```
size= 5  contents= 10 10 10 10 10
```

```
r.resize(8); r[7] = 20;
showVector(r);
```
```
size= 8  contents= 10 10 10 10 10 0 0 20
```

```
r.resize(3);
showVector(r);
```
```
size= 3  contents= 10 10 10
```

```
r.reserve(100);
cout << r.size() << " " << r.capacity() << endl;
```
```
3 100
```

Containers: Λίστα

- Τύπος `list<T>`
  - διπλά συνδεδεμένη λίστα, γραμμική διάσχιση
  - εισαγωγή και αφαίρεση σε αρχή, τέλος
  - `front`, `back`, `empty`
  - `push_front`, `pop_front`
  - `push_back`, `pop_back`
  - εισαγωγή και διαγραφή ενδιάμεσα
  - `insert`, `erase`

Containers: Λίστα

```
void showList(const list<int> &l) {
  list<int>::iterator it;
  cout << "size= " << l.size() << "  contents= ";
  for (it = l.begin(); it!= l.end(); ++it) cout << *it << " ";
  cout << endl;
}

list<int> l;
for (int i=1; i<5; i++) l.push_back(i);
for (int i=0; i<5; i++) l.push_front(-i);
showList(l);
```

```
size= 9  contents= -4 -3 -2 -1 0 1 2 3 4
```

```
for (int i=1; i<5; i++) {
  cout << l.front() << " " << l.back() << " ";
  l.pop_front(); l.pop_back();
}
showList(l);
```

```
-4 4 -3 3 -2 2 -1 1
```

```
size= 1  contents= 0
```

Containers: Λίστα

```
size= 11  contents= 0 1 2 3 4 5 9 8 7 6 5
```

```
l.reverse();
showList(l);
```

```
size= 11  contents= 5 6 7 8 9 5 4 3 2 1 0
```

```
l.sort();
showList(l);
```

```
size= 11  contents= 0 1 2 3 4 5 5 6 7 8 9
```

```
l.push_back(5);
showList(l);
```

```
size= 12  contents= 0 1 2 3 4 5 5 6 7 8 9 5
```

```
l.unique();
showList(l);
```

```
size= 12  contents= 0 1 2 3 4 5 6 7 8 9 5
```

- Τύπος **`forward_list<T>`**
  - απλά συνδεδεμένη λίστα
- Τύπος **`deque<T>`**
  - όπως το vector αλλά με αποδοτική εισαγωγή και αφαίρεση στην αρχή και στο τέλος

STL και templates

```
template <typename T>
void showAnything(const T &iterable) {
  cout << "size= " << iterable.size() << "  contents= ";
  typename T::iterator it;
  for (it = iterable.begin(); it != iterable.end(); ++it)
    cout << *it << " ";
  cout << endl;
}
```

Ε: Τι απαιτείται για να λειτουργεί η **showAnything;**

Α: - το **T** να είναι iterable object
- να έχει οριστεί (αν απαιτείται) ο τελεστής **<<**

```
template <typename T>
void showAnything(const T &iterable) {
  cout << "size= " << iterable.size() << "  contents= ";
  for (auto element : iterable) cout << element << " ";
  cout << endl;
}
```

Στοίβα και ουρά

- Τύπος **`stack<T>`**
  - container adapter, όχι container
  - default container: **`deque<T>`**
  - εισαγωγή και αφαίρεση στην κορυφή
  - push, pop, top, empty

```
stack<int> s;
for (int i=0; i<5; i++) s.push(i*10);
while (!s.empty()) {
  cout << s.top() << " ";
  s.pop();
}
```

```
40 30 20 10 0
```

Στοίβα και ουρά

- Τύπος **`queue<T>`**
  - container adapter, όχι container
  - εισαγωγή στο τέλος, αφαίρεση από την αρχή
  - **`push, pop, front, empty`**

- Στοίβες και ουρές μπορούν να υλοποιηθούν με χρήση διαφορετικών containers, πχ:

```
stack<int, vector<int>> s;
```

Τύπος που περιέχει το stack

Container που υλοποιεί το stack

Στοίβα και ουρά

```
void showQueue(queue<int> q) {
  while (!q.empty()) {
    cout << q.front() << " ";
    q.pop();
  }
}

queue<int> q;
for (int i=1; i<=5; i++)
  q.push(i*10);
showQueue(q);
while (!q.empty()) {
  q.pop();
  showQueue(q);
}
```

```
10 20 30 40 50
20 30 40 50
30 40 50
40 50
50
```

Ζεύγη

- Τύπος `pair<T1, T2>`
  - `make_pair`, `first`, `second`

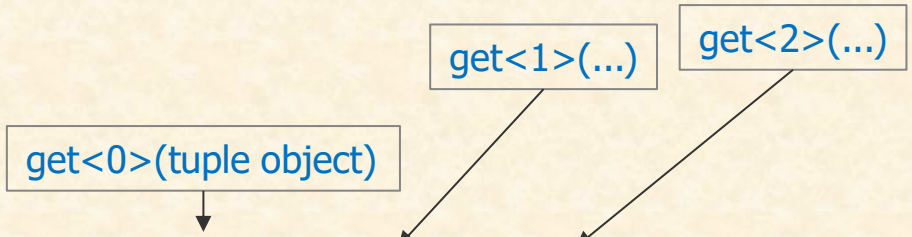
```
#include <utility>
double measure(const pair<double, double> p) {
  return sqrt(p.first*p.first + p.second*p.second);
}

int main() {
  pair<double, double> c1(3, 4);
  pair<double, double> c2;
  c2 = make_pair(1, 1);
  cout << measure(c1) << "  " << measure(c2) << endl;
}
```

Ζεύγη - χρήση

- Pair – πότε μου είναι χρήσιμο
  - Συνδυάζω 2 αντικείμενα, ιδίως όταν δεν έχει νόημα να συγχωνευτούν
  - Επιστρέφω 2 τιμές από μία συνάρτηση
  - Ετοιμοι constructors, operator<
  - ...
  - Προτίμηση, τεχνική, στυλ, κλπ

Πλειάδες (tuples) στη C++11

- Τύπος **`tuple<T1, ... Tn>`**
  - **`make_tuple`, `get`**




```
#include <tuple>

...

void f(const tuple<string, int, bool> &t) {
  if (get<2>(t)) cout << get<0>(t) << endl;
  else           cout << get<1>(t) << endl;
}

tuple<string, int, bool> t("Hi", 1, true);

f(t); f(make_tuple("No", 42, false));
```

Πλειάδες (tuples) στη C++11

```
// dates are: day, month, year
typedef tuple<int, string, int> myDate;
using myDate = tuple<int, string, int>;

void printdate(const myDate &d, const string &dateFormat) {
  if (dateFormat == "us")
    cout << get<1>(d) << " " << get<0>(d)
         <<", " << get<2>(d) << endl;
  else
    cout << get<0>(d) << " " << get<1>(d)
         <<" " << get<2>(d) << endl;
}
...
myDate easter22, easter23(16, "April", 2023);
easter22 = make_tuple(22,"April", 2022);
printdate(easter22, "us");
printdate(easter23, "eu");
```

Καλή πρακτική!

Εναλλακτικά

```
April 22, 2022
16 April 2023
```

Πλειάδες - χρήση

- Tuple – πότε μου είναι χρήσιμη
  - Συνδυάζω N αντικείμενα, ιδίως όταν δεν έχει νόημα να συγχωνευτούν
  - Επιστρέφω N τιμές από μία συνάρτηση
  - Ετοιμοι constructors, operator<
  - ...
  - Προτίμηση, τεχνική, στυλ, κλπ

  - Προσοχή:
    ο operator< δεν είναι πάντα αυτό που θέλω!

Πλειάδες - χρήση

```
typedef tuple<int, string, int> myDate;  // day, month, year

void printDate(const myDate &d, const string &dateFormat) {
  if (dateFormat == "us")
    cout << get<1>(d) << " " << get<0>(d) << ", " << get<2>(d) << endl;
  else
    cout << get<0>(d) << " " << get<1>(d) << " " << get<2>(d) << endl;
}

void tuple_demo() {
  myDate easter22, easter23(16, "April", 2023);
  easter22 = make_tuple(22, "April", 2022);

  printDate(easter22, "us");
  printDate(easter23, "eu");

  if (easter23 < easter22)
    cout << "something is wrong with time..." << endl;
  else
    cout << "time moves on!" << endl;
}
```

Πλειάδες - χρήση

- Υλοποίηση της σύγκρισης (operator<)

```
typedef tuple<int, string, int> myDate;  // day, month, year
// BAD implementation...
bool operator<(const myDate &a, const myDate &b) {
  int m1, m2;
  if (get<1>(a) == "January") m1 = 1;
  else if (get<1>(a) == "February") m1 = 2;
  (...else)
  else if (get<1>(a) == "December") m1 = 12; else m1 = 0;
  if (get<1>(b) == "January") m2 = 1;
  else if (get<1>(b) == "February") m2 = 2;
  (...else)
  else if (get<1>(b) == "December") m2 = 12; else m2 = 0;
  int val1 = get<2>(a)*365 + m1 * 30 + get<0>(a);
  int val2 = get<2>(b)*365 + m2 * 30 + get<0>(b);
  return val1 < val2;
}
```

Πλειάδες - χρήση

```
typedef tuple<int, string, int> myDate;  // day, month, year
// BETTER implementation...
bool operator<(const myDate &a, const myDate &b) {
  string allMonths = "January**February*March****
                      April****May******June*****July*****August***
                      September*October**November*December*";
  int month_a = allMonths.find(get<1>(a)) / 9 + 1;
  int month_b = allMonths.find(get<1>(b)) / 9 + 1;
  int date1value = get<2>(a)*365 + month_a * 30 + get<0>(a);
  int date2value = get<2>(b)*365 + month_b * 30 + get<0>(b);
  return date1value < date2value;
}
```

length=9 length=9 length=9

length=9

Πλειάδες - χρήση

```
typedef tuple<int, string, int> myDate;  // day, month, year

// EVEN BETTER implementation...
string canonical(const string &month) {
  if (month.length() < 3) throw domain_error("invalid month");
  string result;
  transform(month.begin(), month.begin() + 3,
            result.begin(), ::tolower);
  result[0] = ::toupper(result[0]);
  return result;
}

int numerical_month(const string &month) {
  static const string months = "JanFebMarAprMayJunJulAugSepOctNovDec";
  int result = months.find(canonical(month));
  if (result == -1) throw domain_error("invalid month");
  return 1 + result / 3;
}

bool operator<(const myDate &a, const myDate &b) {
  return make_tuple(get<2>(a), numerical_month(get<1>(a)), get<0>(a))
       < make_tuple(get<2>(b), numerical_month(get<1>(b)), get<0>(b));
}
```

Πλειάδες - χρήση

```
vector<myDate> someDates = {
  make_tuple(16, "April", 2022),
  make_tuple(15, "April", 2022),
  make_tuple(14, "April", 2023),
  make_tuple(15, "May", 2022),
  make_tuple(18, "January", 2020)
};

bool dateComparison(const myDate &a, const myDate &b) {
  return a < b;
}

sort(someDates.begin(), someDates.end(), dateComparison);

for (const myDate &d : someDates)
  printDate(d, "us");
```

```
January 18, 2020
April 15, 2022
May 15, 2022
April 16, 2022
April 14, 2023
```

(Θα επανέλθουμε, στη συζήτηση για STL algorithms)

Σύνολο

- Τύπος **`set<T>`**
  - μοναδικά στοιχεία (αλλιώς **`multiset<T>`**)
  - υλοποίηση με ισοζυγισμένο δένδρο δυαδικής αναζήτησης
  - εισαγωγή, εύρεση και αφαίρεση σε O(log*n*)
  - **`insert`**, **`erase`**, **`find`**, **`size`**
  - πράξεις συνόλων (υλοποιούνται και για άλλους containers, λιγότερο αποδοτικά): **`includes`**, **`set_union`**, **`set_intersection`**, **`set_difference`**

Σύνολο

```
#include <set>
...
set<int> s;
while (true) {
  int x;
  cin >> x;
  if (s.find(x) == s.end())
    s.insert(x);
  else
    cout << "You lose, I've seen" << x
         << "before!" << endl;
}
```

Σύνολο

```
set<int> someInts= {10, 15, 3, 19, 2, 67, 69, 68};
for (int i : someInts) cout << i << " "; cout << endl;
```

```
2 3 10 15 19 67 68 69
```

```
someInts.insert(11);
for (int i : someInts) cout << i << " "; cout << endl;
```

```
2 3 10 11 15 19 67 68 69
```

```
set<int> moreInts= {70, 8, 81, 5, 10, 69, 67, 31, 3};
set<int> unionDemo;
set_union(someInts.begin(), someInts.end(),
          moreInts.begin(), moreInts.end(),
             inserter(unionDemo, unionDemo.begin()));

for (int i : unionDemo) cout << i << " "; cout << endl;
```

```
2 3 5 8 10 11 15 19 31 67 68 69 70 81
```

Σύνολο

```
set<int> someInts= {10, 15, 3, 19, 2, 67, 69, 11, 68};

set<int> moreInts= {70, 8, 81, 5, 10, 69, 67, 31, 3};


set<int> intersectionDemo;
set_intersection(someInts.begin(), someInts.end(),
                 moreInts.begin(), moreInts.end(),
                 inserter(intersectionDemo,
                          intersectionDemo.begin()));

for (int i : intersectionDemo) cout << i << " ";
cout << endl;
```

```
3 10 67 69
```

Σύνολο: παράδειγμα (set)

```
struct data {
  int number;
  string s;
};

bool operator<(const data &d1, const data &d2) {
  return d1.number < d2.number;
}

void set_demo() {
  set<data> mySet;
  int n;
  cin >> n;
  for (int i=0; i<n; ++i) {
    data d;
    cin >> d.number >> d.s;
    mySet.insert(d);
  }
  cout << "mySet contents\n";
  for (auto const &item : mySet)
    cout << item.number << " " << item.s << endl;
}
```

Δοκιμάστε να δώσετε δύο ίδιους αριθμούς με διαφορετικές συμβολοσειρές!


```
mySet contents
1 one
2 one
3 two
4 four
5 four
6 six
7 six
8 seven
9 nine
10 nine
```

Τυπική διάτρεξη: iterators

```
struct data {
  int number;
  string s;
};

void set_demo() {
  set<data> mySet;
  ...
  set<data>::iterator it;
  for (it = mySet.begin(); it != mySet.end(); ++it) {
    cout << it->number << " " << it->s << endl;
  }
}
```

set<data>:
Ο τύπος των δεδομένων του container που διατρέχουμε

mySet:
Ο container που διατρέχουμε

begin(), end(), ++:
Χρήση του iterator

Δήλωση iterator:
Ορίζει iterator με τον οποίο διατρέχουμε τον container
Τύπος **container::iterator**

Χρήση iterator:
Η μεταβλητή iterator λειτουργεί σαν δείκτης

Με τον τρόπο αυτό διατρέχουμε οποιοδήποτε τμήμα του container

Εύκολη διάτρεξη: auto

```
struct data {
   int number;
   string s;
};

void set_demo() {
   set<data> mySet;
   ...
   for (auto const &item : mySet) {
      cout << item.number << " " << item.s << endl;
   }
}
```

set<data>:
Ο τύπος των δεδομένων του container που διατρέχουμε

mySet:
Ο container που διατρέχουμε

auto:
Ορίζει αυτόματα τον τύπο της μεταβλητής με την οποία διατρέχουμε τον container

item:
Μεταβλητή που παίρνει τιμές από τα περιεχόμενα του container που διατρέχουμε

Με τον τρόπο αυτό διατρέχουμε ολόκληρο τον container

Σύνολο: παράδειγμα (set)

```
struct data {
  int number;
  string s;
};

bool operator<(const data &d1, const data &d2) {
  if (d1.s != d2.s) return d1.s < d2.s;
  return d1.number < d2.number;
}

void set_demo() {
  set<data> mySet;
  int n;
  cin >> n;
  for (int i=0; i<n; ++i) {
    data d;
    cin >> d.number >> d.s;
    mySet.insert(d);
  }
  cout << "mySet contents\n";
  for (auto const &item : mySet)
    cout << item.number << " " << item.s << endl;
}
```

Εναλλακτική υλοποίηση του <

Πολυσύνολο: παράδειγμα (multiset)

```
struct data {
  int number;
  string s;
};

bool operator<(const data &d1, const data &d2) {
  if (d1.s != d2.s) return d1.s < d2.s;
  return d1.number < d2.number;
}

void multiset_demo() {
  multiset<data> myMultiSet;
  int n;
  cin >> n;
  for (int i=0; i<n; ++i) {
    data d;
    cin >> d.number >> d.s ;
    myMultiSet.insert(d); myMultiSet.insert(d);
  }
  cout << "myMultiSet contents\n";
  for (auto const & item: myMultiSet)
    cout << item.number << " " << item.s << endl;
}
```

```
myMultiSet contents
1 one
1 one
2 two
2 two
3 three
3 three
4 four
4 four
5 five
5 five
6 six
6 six
7 seven
7 seven
9 nine
9 nine
```

Πολυσύνολο: παράδειγμα (λάθος update)

```
multiset<data> mySet;
int n;
cin >> n;
for (int i=0; i<n; ++i) {
  data d;
  cin >> d.number >> d.s ;
  mySet.insert(d);
}
cout << "mySet initial contents" << endl;
for (auto item : mySet )
  cout << "{ " << item.number << ", " << item.s << "} ";
// modifying contents
for (auto item : mySet) {
  item.number *= 2; item.s += " upd";
}
cout << endl << "mySet x2" << endl;
for (auto item : mySet)
  cout << "{ " << item.number << ", " << item.s << "} ";
```

Δεν αλλάζει τίποτα!
Το item είναι αντίγραφο!

```
mySet initial contents
{ 1, one} { 2, two} { 3, three} { 4, four} { 5, five} { 6, six} { 7, seven} { 9, nine}
mySet x2
{ 1, one} { 2, two} { 3, three} { 4, four} { 5, five} { 6, six} { 7, seven} { 9, nine}
```

Πολυσύνολο: παράδειγμα (λάθος update[2])

```
multiset<data> mySet;
int n;
cin >> n;
for (int i=0; i<n; ++i) {
  data d;
  cin >> d.number >> d.s ;
  mySet.insert(d);
}
cout << "mySet initial contents" << endl;
for (auto item : mySet )
  cout << "{ " << item.number << ", " << item.s << "} ";
// modifying contents
for (auto & item : mySet) {
  item.number *= 2; item.s += " upd";
}
cout << endl << "mySet x2" << endl;
for (auto item : mySet)
  cout << "{ " << item.number << ", " << item.s << "} ";
```

Δεν κάνει compile!
Το item& είναι const!

```
mySet initial contents
{ 1, one} { 2, two} { 3, three} { 4, four} { 5, five} { 6, six} { 7, seven} { 9, nine}
mySet x2
{ 1, one} { 2, two} { 3, three} { 4, four} { 5, five} { 6, six} { 7, seven} { 9, nine}
```

Πολυσύνολο: παράδειγμα (update! σχεδόν)

```
multiset<data> mySet;
int n;
cin >> n;
for (int i=0; i<n; ++i) {
  data d;
  cin >> d.number >> d.s ;
  mySet.insert(d);
}
cout << "mySet initial contents" << endl;
for (auto item:  mySet )
  cout << "{ " << item.number << ", " << item.s << "} ";
// modifying contents
multiset<data> myNewSet;
{ for (auto item : mySet)
    myNewSet.insert({item.number * 2, item.s + " upd"});
  mySet = myNewSet;
}
cout << endl << "mySet x2" << endl;
for (auto item : mySet)
  cout << "{ " << item.number << ", " << item.s << "} ";
```

Αυτό λειτουργεί.
Φτιάχνει όμως καινούργιο multiset!

```
mySet initial contents
{ 1, one} { 2, two} { 3, three} { 4, four} { 5, five} { 6, six} { 7, seven} { 9, nine}
mySet x2
{ 2, one upd} { 4, two upd} { 6, three upd} { 8, four upd} { 10, five upd} { 12, six upd} { 14, seven upd} { 18, nine upd}
```

Παρόμοιο παράδειγμα update σε vector

```
vector<data> myVector;
int n;
cin >> n;
for (int i=0; i<n; ++i) {
  data d;
  cin >> d.number >> d.s ;
  myVector.push_back(d);
}
cout << "myVector (std::vector) initial contents" << endl;
for (auto item:  myVector )
  cout << item.number << " " << item.s << endl;
// you need to use reference here
for (auto & item:  myVector ) {
  item.number *=2;
  item.s += " upd";
}
cout << "myVector (std::vector) x2" << endl;
for (auto item: myVector)
  cout << item.number << " " << item.s << endl;
```

Κάνει compile και λειτουργεί!
Το item& **δεν** είναι const!

Χάρτης

- Τύπος **`map<K, T>`**
  - απεικόνιση κλειδιών τύπου **`K`** σε τιμές τύπου **`T`**
  - μοναδικές τιμές κλειδιών (αλλιώς **`multimap<K, T>`**)
  - υλοποίηση με ισοζυγισμένο δένδρο δυαδικής αναζήτησης
  - εισαγωγή, εύρεση και αφαίρεση σε O(logn)
  - **`insert, erase, find, size`**
  - **`operator[]`**

Χάρτης

```
#include <map>

map<int, string> m;
m.insert(make_pair(1, "one"));
m.insert(make_pair(2, "two"));
m.insert(make_pair(3, "three"));

int x;
cout << "what? "; cin >> x;

map<int, string>::iterator p;

p = m.find(x);

if (p != m.end())
   cout << p->second << endl;
```

Συνήθως όμως, τα maps δε χρησιμοποιούνται έτσι!

Χάρτης

```
map<int, string> m;
m[1] ="one";
m[2] ="two";
m[3] ="three";

int x;
cin >> x;
if (!m[x].empty())
  cout << m[x] << endl;

for (x=0; x<5; x++)
  cout << x << " -> ("
       << m[x] << ")" << endl;
```

Αλλά έτσι!

```
what? 2
two
0 ->  ()
1 ->  (one)
2 ->  (two)
3 ->  (three)
4 ->  ()
```

Προσοχή!
Αν δεν υπάρχει το m[x] τότε η αναφορά σε αυτό θα το δημιουργήσει με τιμή ένα κενό string

Χάρτης

```
map<string, string> m;
string en[] = {"one", "two", "three", "four"};
string it[] = {"uno", "due", "tre", "quattro"};

for (int i=0; i < 4; i++)
  m[en[i]] = it[i];
```

```
m["one"] = "uno"
m["two"] = "due"
m["three"] = "tre"
m["four"] = "quattro"
```

```
for (string n:en)
  cout << n << " ->  (" << m[n] << ")" << endl;
```

```
one ->  (uno)
two ->  (due)
three ->  (tre)
four ->  (quattro)
```

Χάρτης

$O(2^n)$

```
int fib(int n) {
  return n < 2 ? n : fib(n-1) + fib(n-2);
}

int memo_fib(int n) {
  static map<int, int> cache;
  // Αν είναι εύκολο, τελείωνε!
  if (n < 2) return n;
  // Αν υπάρχει ήδη, μην το υπολογίσεις.
  map<int, int>::iterator p = cache.find(n);
  if (p != cache.end()) return p->second;
  // Αλλιώς υπολόγισε και αποθήκευσέ το.
  int v = memo_fib(n-1) + memo_fib(n-2);
  cache[n] = v;  return v;
}
```

Memoization

$O(n \log n)$

Χρήσιμοι αλγόριθμοι στην STL

- Αναζήτηση
- Ταξινόμηση
- Λεξικογραφική διάταξη
- Γεννήτρια μεταθέσεων
- κ.λπ.

```
#include <algorithm>
```

Αναζήτηση (i)

- `find(first, last, value)`
  - σε χρόνο $O(n)$

- `binary_search(first, last, value)`
- `binary_search(first, last, value, compare)`
  - σε χρόνο $O(\log n)$

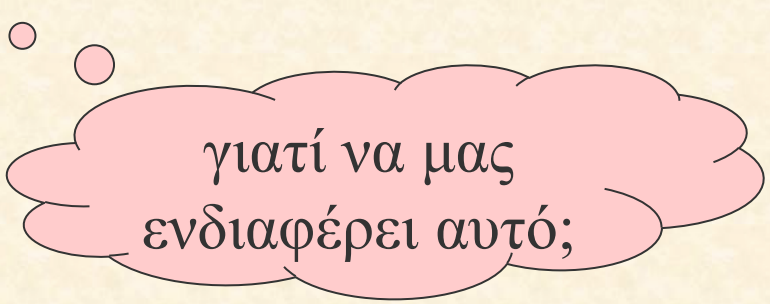
Αναζήτηση (ii)

```
vector<int> v;
...
vector<int>::iterator i =
  find(v.begin(), v.end(), 42);
if (i == v.end())
  cout << "not found" << endl;

// Αν οι αριθμοί είναι ταξινομημένοι:
vector<int>::iterator i =
  binary_search(v.begin(), v.end(), 42);
if (i != v.end())
  *i = 17;              // Άλλαξε το 42 σε 17
```

Ταξινόμηση (i)

- `sort(first, last)`
- `sort(first, last, compare)`
  - σε χρόνο $O(n \log n)$ στη μέση περίπτωση

- `stable_sort(first, last)`
- `stable_sort(first, last, compare)`
  - σε χρόνο $O(n \log n)$ στη χειρότερη περίπτωση
  - διατηρεί τη σειρά ίσων στοιχείων



Ταξινόμηση (ii)

```
vector<int> v;
...
// Σε αύξουσα σειρά:
sort(v.begin(), v.end());

// Σε φθίνουσα σειρά:
sort(v.begin(), v.end(), greater<int>());
```

- Το `greater<int>()` κατασκευάζει ένα **function object**. Χρειάζεται:

```
#include <functional>
```

Ταξινόμηση (iii)

```
bool my_less_than(int x, int y) {
  return abs(x) < abs(y);
}

struct my_comparator {
  bool operator()(int x, int y) {
    return abs(x) < abs(y);
  }
};

vector<int> v;
...
// Σε αύξουσα σειρά κατ' απόλυτο τιμή, εναλλακτικά:
sort(v.begin(), v.end(), my_less_than);
sort(v.begin(), v.end(), my_comparator());
```

Ταξινόμηση (iv)

- **`partial_sort(first, middle, last)`**
- **`partial_sort(first, middle, last, compare)`**
  - σε χρόνο $O(n \log m)$ στη χειρότερη περίπτωση (n = last–first, m = middle–first)
  - ταξινομεί μόνο τα m πρώτα στοιχεία

```
vector<int> v;
...
partial_sort(v.begin(), v.begin() + 10,
             v.end());
```

Ταξινόμηση (v)

- **`nth_element(first, nth, last)`**
- **`nth_element(first, nth, last, compare)`**
  - σε χρόνο O(*n*) στη μέση περίπτωση
  - το ζητούμενο στοιχείο θα είναι στη θέση του
  - τα προηγούμενά του θα είναι μικρότερα, τα επόμενά του θα είναι μεγαλύτερα

```
vector<int> v;
...
nth_element(v.begin(), v.begin() + 10,
            v.end());
```

Εύρεση μικρότερου/μεγαλύτερου

- `min_element(first, last)`
- `min_element(first, last, compare)`
- `max_element(first, last)`
- `max_element(first, last, compare)`
  - σε χρόνο $O(n)$