ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΤΕΧΝΙΚΕΣ 20256

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

Διδάσκοντες:

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

progtech@courses.softlab.ntua.gr

27/3/2026

Διαφάνειες παρουσιάσεων

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

Ζεύγη

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

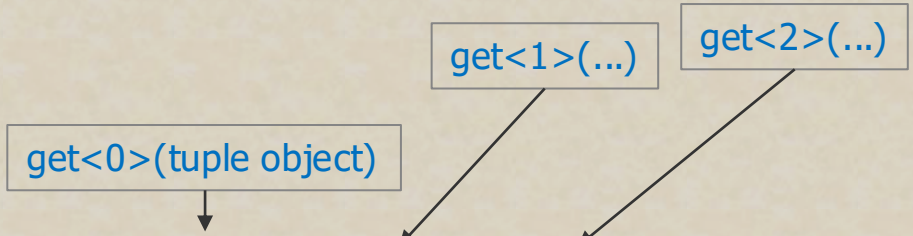
```
#include <utility>
double measure(const pair<double, double> p) {
  return sqrt(p.first*p.first + p.second*p.second);
}

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***
                      SeptemberOctober**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

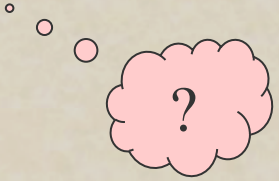
```
set<int> someInts= {10, 15, 3, 19, 2, 67, 69, 11, 68};

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

Συνήθως όμως, τα maps δε χρησιμοποιούνται έτσι!

```
#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;
```

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!

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

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

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 $O(n)$

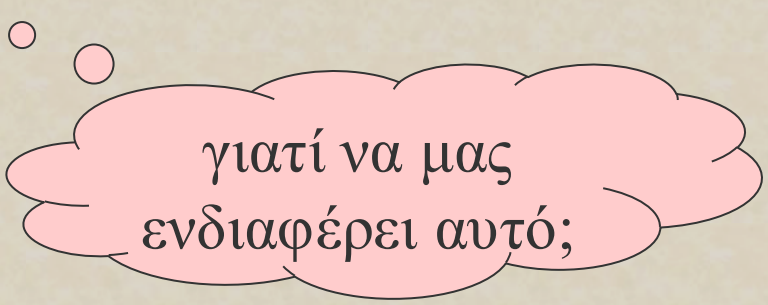
- `binary_search(first, last, value)`
- `binary_search(first, last, value, compare)`
  - σε χρόνο $O(\log n)$

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 $O(n)$

ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΤΕΧΝΙΚΕΣ 20256

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

Διδάσκοντες:

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

progtech@courses.softlab.ntua.gr

27/3/2026

Διαφάνειες παρουσιάσεων

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

sierra tango lima echo x-ray alpha mike papa lima echo

Παραδείγματα STL

Παράδειγμα STL #1

- Μετατροπή από/προς το φωνητικό αλφάβητο

Παράδειγμα STL #1 - maps

- Υλοποίηση με map
  - string char2phonetic
  - string string2phonetic
  - char phonetic2char
  - string phonetic2word (memoization, too)

```
string char2phonetic(char)
```

'a' → "alpha",
'b' → "bravo"

```
string string2phonetic(string)
```

"ece" →
"echo charlie echo"

```
char phonetic2char(string)
```

"alpha" → 'a',
"beta" → 'b'

```
string phonetic2word(string)
```

"echo charlie echo" →
"ece"

Παράδειγμα STL #1 - maps

```
char char2phonetic (const char &c) {
  static const string phoneticWords[] = {
    "alpha", "bravo", "charlie", "delta", "echo", "foxtrot",
    "golf", "hotel", "india", "juliett", "kilo", "lima",
    "mike", "november", "oscar", "papa", "quebec", "romeo",
    "sierra", "tango", "uniform", "victor", "whiskey", "x-ray",
    "yankee", "zulu", " ", "@", "."
  };
  static map<char, string> phonetic;
  // Setup the mapping only once!
  if (phonetic.empty()) {
    for (const string &w : phoneticWords)
      phonetic[w.front()] = w;
  }
  // Do the job!
  if (phonetic.find(c) == phonetic.end()) return "?*?*?";
  return phonetic[c];
}
```

Παράδειγμα STL #1 - maps

```
string string2phonetic(const string &ss) {
  string result = "";
  for (char cc : ss) result += char2phonetic(cc) + " ";
  return result.substr(0, result.length()-1);
}

// Use memoization, if you expect to see the same word many times!
string string2phonetic_better(const string &ss) {
  static map<string, string> cachedWords;

  // search for known word
  map<string, string>::iterator cw = cachedWords.find(ss);
  if (cw != cachedWords.end()) return cw->second;

  // word not known
  string result = "";
  for (char cc : ss) result += char2phonetic(cc) + " ";
  cachedWords[ss] = result.substr(0, result.length()-1);
  return result;
}
```

Παράδειγμα STL #1 - maps

```
char phonetic2char(const string &searchString) {
  static const string phoneticWords[] = {
    "alpha", "bravo", "charlie", "delta", "echo", "foxtrot",
    "golf", "hotel", "india", "juliett", "kilo", "lima",
    "mike", "november", "oscar", "papa", "quebec", "romeo",
    "sierra", "tango", "uniform", "victor", "whiskey", "x-ray",
    "yankee", "zulu", " ", "@", "."
  };
  static map<string, char> invPhonetic;
  // Setup the mapping only once!
  if (invPhonetic.empty()) {
    for (const string &w : phoneticWords)
      invPhonetic[w] = w.front();
  }
  // Do the job!
  map<string, char>::iterator i = invPhonetic.find(searchString);
  if (i == invPhonetic.end()) return '?';
  return i->second;
}
```

Παράδειγμα STL #1 - maps

```
string phonetic2word(const string &src) {
  string result = "";
  string oneWord;
  istringstream stringStream(src);
  while (stringStream >> oneWord)
    result += phonetic2char(oneWord);
  return result;
}
```

Υλοποίηση χωρίς κλάσεις

```
string originalText = "";
string oneWord, originalWord;

while(inFile >> inWord) {
  string p = string2phonetic (inWord);
  cout << p << " " << endl;

  originalWord = phonetic2word(p);
  originalText += originalWord + " / ";
}
```

Παράδειγμα STL #1 – maps με κλάσεις

- Σχεδίαση κλάσης **readable**

```
class readable {
private:
  string normalText; string phoneticText;
  static map<char, string> phoneticMap;
  static map<string, char> invPhoneticMap;
  static const string phoneticWords[29];
  static void initializeMappings();
  static string char2phonetic(const char &c);              // direct
  static string text2phonetic(const string &txt);
  static char phonetic2char(const string &searchString); // reverse
  static string phonetic2text(const string &t);
public:
  readable();
  readable(const string &sometext);
  string phonetic() const;                    // getter
  string text() const;                        // getter
  void setPhonetic(const string &pText);  // setter
  void setText(const string &nText);       // setter
};
```

Αφαίρεση και δομές δεδομένων

- Οι λεπτομέρειες υλοποίησης επηρεάζουν την πολυπλοκότητα των αλγορίθμων και τη χρήση πόρων γενικά
- Η χρήση ΑΤΔ ελαχιστοποιεί τις αλλαγές που απαιτούνται σε ένα πρόγραμμα που χρησιμοποιεί έναν ΑΤΔ, όταν αλλάξει ο τρόπος υλοποίησης

Παράδειγμα STL #1 – maps με κλάσεις

```
const string readable::phoneticWords[29] = {
        "alpha", "bravo", "charlie", "delta", "echo", "foxtrot",
        "golf", "hotel", "india", "juliett", "kilo", "lima",
        "mike", "november", "oscar", "papa", "quebec", "romeo",
        "sierra", "tango", "uniform", "victor", "whiskey", "x-ray",
        "yankee", "zulu", " ", "@", "."};

readable::readable() : normalText(""), phoneticText("") {
  initializeMappings();
}

readable::readable(const string &sometext) : normalText(sometext) {
  initializeMappings();
  phoneticText = text2phonetic(sometext);
}

void initializeMappings() {
  // Initialize the mappings only once!
  if (!phoneticMap.empty()) return;
  for (const string &word : phoneticWords) {
    phoneticMap[word.front()] = word;
    invPhoneticMap[word] = word.front();
  }
}
```

Παράδειγμα STL #1 – maps με κλάσεις

```
string readable::char2phonetic(const char &c) {
  if (phoneticMap.find(c) == phoneticMap.end()) return "?";
  return phoneticMap[c];
}

string readable::text2phonetic(const string &txt) {
  string result = "";
  for (char c : txt) result += char2phonetic(c) + " ";
  return result.substr(0, result.length()-1);
}

char readable::phonetic2char(const string &searchString) {
  auto ipi = invPhoneticMap.find(searchString);
  return ipi == invPhoneticMap.end()? '?' : ipi->second;
}

string readable::phonetic2text(const string &t) {
  string w = "", result = "";
  istringstream stringStream(t);
  while (stringStream >> w) result += phonetic2char(w);
  return result;
}
```

Παράδειγμα STL #1 – maps με κλάσεις

```
string readable::phonetic() { return phoneticText; }
string readable::text()     { return normalText; }

void readable::setPhonetic(const string & pText) {
  phoneticText = pText;
  normalText   = phonetic2text(pText);
}

void readable::setText(const string &nText)  {
  normalText = nText;
  phoneticText = text2phonetic(nText);
}

// ...
readable myRR("treis laloun kai dyo chorevoun (paroimia)");
cout << myRR.text() << endl << myRR.phonetic();
```

treis laloun kai dyo chorevoun (paroimia)
tango romeo echo india sierra lima alpha lima oscar uniform november kilo alpha india delta yankee oscar charlie hotel oscar romeo echo victor oscar uniform november ?*?*? papa alpha romeo oscar india mike india alpha ?*?*?

Αφαίρεση και δομές δεδομένων

- Οι λεπτομέρειες υλοποίησης επηρεάζουν την πολυπλοκότητα των αλγορίθμων και τη χρήση πόρων γενικά
- Η χρήση ΑΤΔ ελαχιστοποιεί τις αλλαγές που απαιτούνται σε ένα πρόγραμμα που χρησιμοποιεί έναν ΑΤΔ, όταν αλλάξει ο τρόπος υλοποίησης

Ιδέα:

Δοκιμάστε να αλλάξετε το **private** τμήμα της κλάσης **readable** χωρίς να αλλάξει τίποτε στο **public**

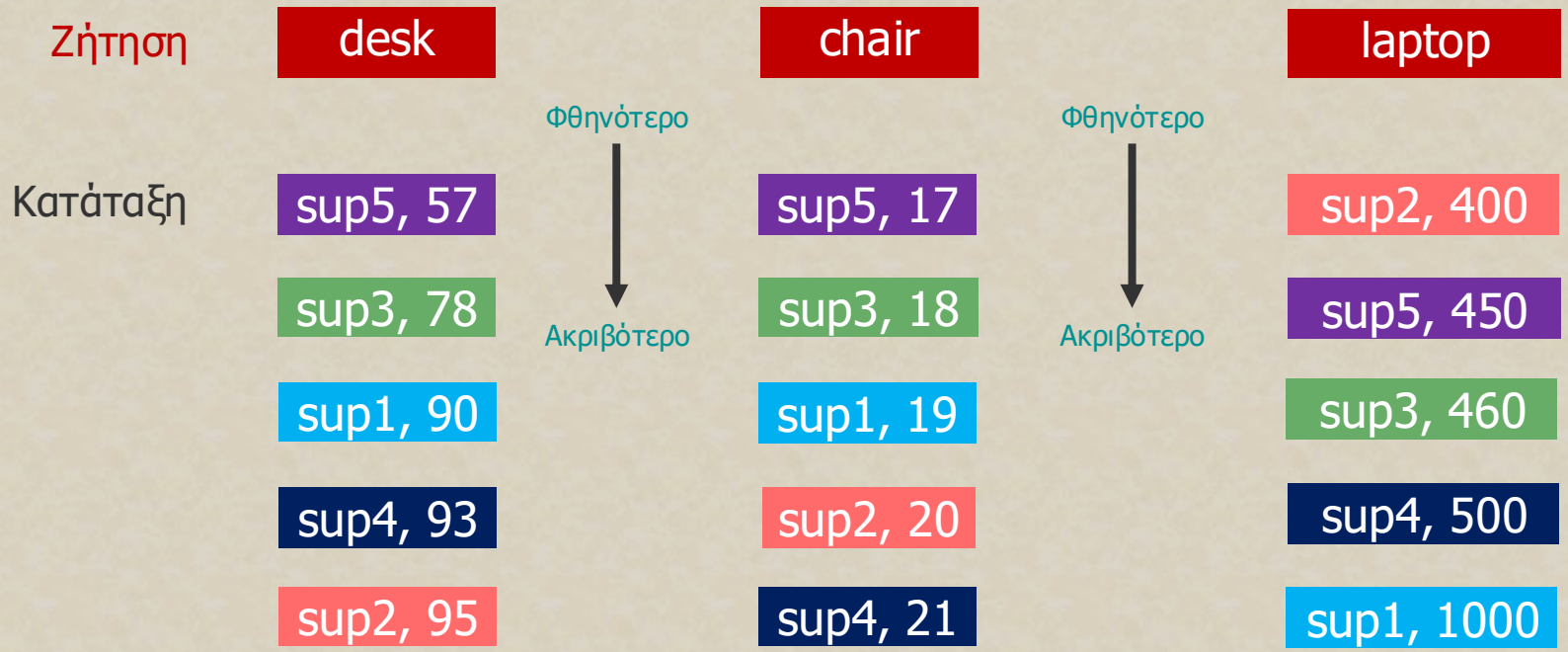
Παράδειγμα #2: Διαγωνισμοί προμηθειών

- Αρχές λειτουργίας
  - Ζήτηση προϊόντων
  - Προσφορές από προμηθευτές (αγορά)
  - Επιλογή προμηθευτή
    - Για κάθε προϊόν επιλέγουμε αυτόν που προσφέρει τη μικρότερη τιμή αγοράς

Παράδειγμα #2: Διαγωνισμοί προμηθειών

Ζήτηση	desk	chair	laptop
Προσφορά	sup1, 90	sup1, 19	sup1, 1000
	sup2, 95	sup2, 20	sup2, 400
	sup3, 78	sup3, 18	sup3, 460
	sup4, 93	sup4, 21	sup4, 500
	sup5, 57	sup5, 17	sup5, 450

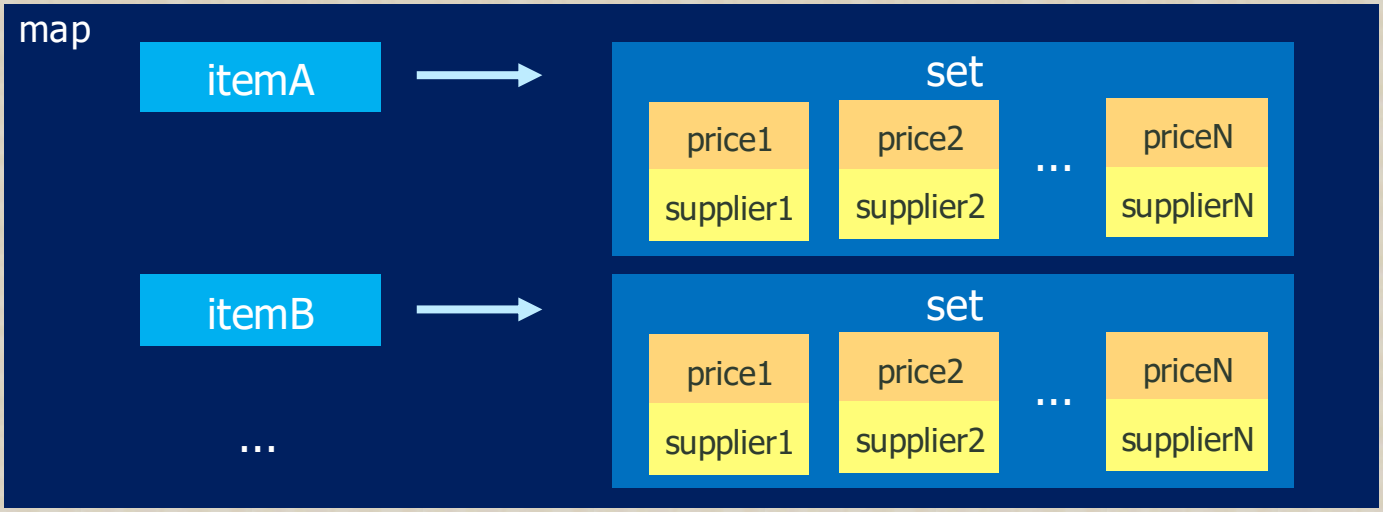
Παράδειγμα #2: Διαγωνισμοί προμηθειών



Συζήτηση: υλοποίηση με STL

- Αντιστοίχιση: Είδος → Προσφορές
  map item(string) set (offers)
  - Περιεχόμενο προσφορών
    - Τιμή μονάδας
    - Προμηθευτής

    struct: price (double), supplier (string)
  - Ταξινόμηση προσφορών: set, με κατάλληλο operator<
    - Τιμή μονάδας

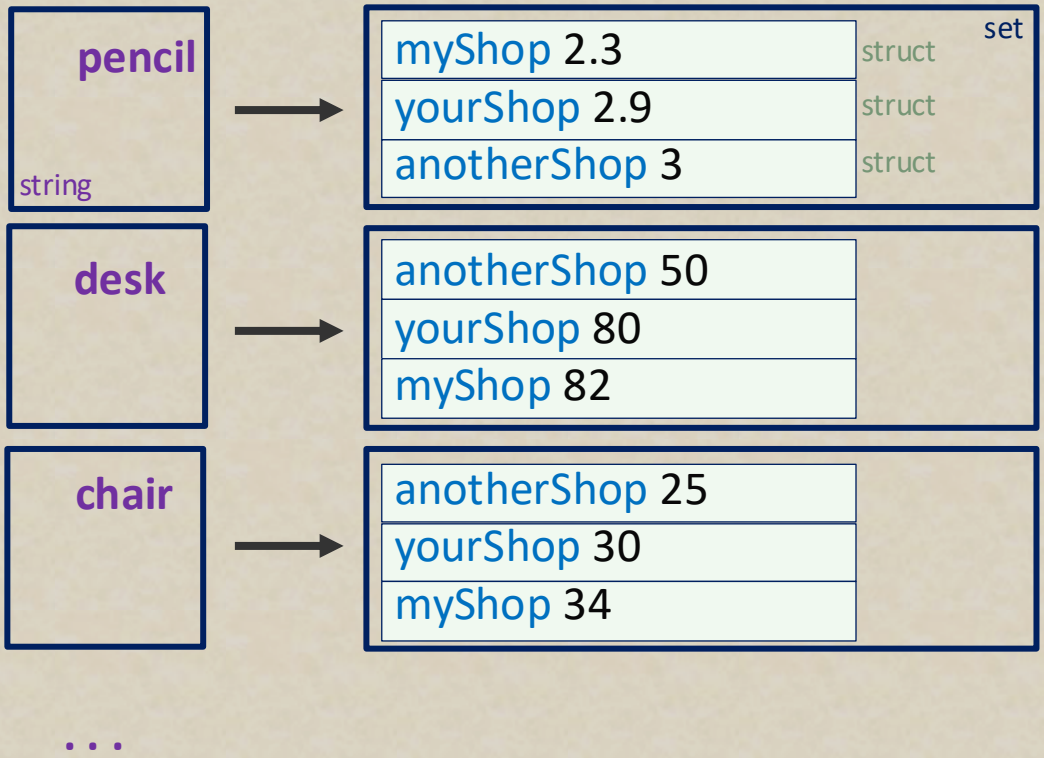


Συζήτηση: υλοποίηση με STL

INPUT

myShop **pencil** 2.3
myShop **desk** 82
myShop **chair** 34
myShop **paper** 2.7
myShop **laptop** 400
yourShop **pencil** 2.9
yourShop **desk** 80
yourShop **chair** 30
yourShop **paper** 3.1
yourShop **laptop** 460
anotherShop **pencil** 3
anotherShop **desk** 50
anotherShop **chair** 25
anotherShop **paper** 3.8
anotherShop **laptop** 550

STL STRUCTURES

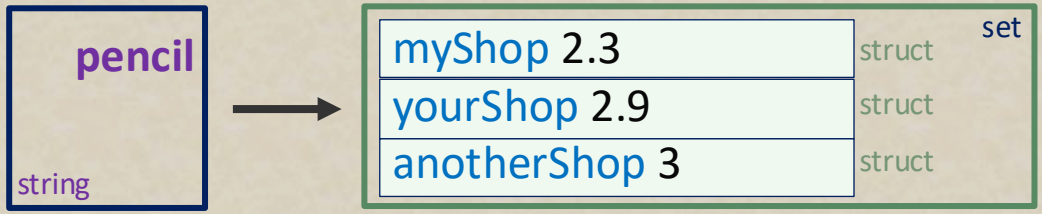


Συζήτηση: υλοποίηση με STL

INPUT

myShop **pencil** 2.3
myShop **desk** 82
myShop **chair** 34
myShop **paper** 2.7
myShop **laptop** 400
yourShop **pencil** 2.9
yourShop **desk** 80
yourShop **chair** 30
yourShop **paper** 3.1
yourShop **laptop** 460
anotherShop **pencil** 3
anotherShop **desk** 50
anotherShop **chair** 25
anotherShop **paper** 3.8
anotherShop **laptop** 550

STL STRUCTURES




```
struct offer {
  string supplier;
  double price;
};
```

```
map<string, set<offer>>
```

```
typedef map<string, set<offer>> bids;
```

Συζήτηση: υλοποίηση με STL

```
struct offer {
  string supplier;
  double price;
};

typedef map<string, set<offer>> bids;

bool operator< (const offer &o1, const offer &o2) {
  if (o1.price != o2.price) return o1.price < o2.price;
  return o1.supplier < o2.supplier;
}

int main() {
  ifstream input("sample_offers.txt");
  string sup, itm;
  double prc;
  bids receivedOffers = {};
  while (input >> sup >> itm >> prc) {
    offer currentOffer = {sup, prc};
    receivedOffers[itm].insert(currentOffer);
  }
  input.close();
  for (bids::iterator iitem=receivedOffers.begin(); iitem!=receivedOffers.end(); ++iitem) {
    cout << iitem->first << ":" << endl;
    for (set<offer>::iterator oo = iitem->second.begin(); oo!=iitem->second.end(); ++oo)
      cout << "    " << oo->supplier << " " << oo->price << endl;
  }
}
```

string → set

empty map

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

Συζήτηση: υλοποίηση με STL

```
struct offer {
  string supplier;
  double price;
};

typedef map<string, set<offer>> bids;

bool operator< (const offer &o1, const offer &o2) {
  if (o1.price != o2.price) return o1.price < o2.price;
  return o1.supplier < o2.supplier;
}

int main() {
  ifstream input("sample_offers.txt");
  string sup, itm;
  double prc;
  bids receivedOffers = {};
  while (input >> sup >> itm >> prc) {
    offer currentOffer = {sup, prc};
    receivedOffers[itm].insert(currentOffer);
  }
  input.close();
  for (const auto &iitem : receivedOffers) {
    cout << iitem.first << ":" << endl;
    for (const auto &oo : iitem.second)
      cout << "    " << oo.supplier << " " << oo.price << endl;
  }
}
```

auto:
Απλούστερη διάτρεξη με auto