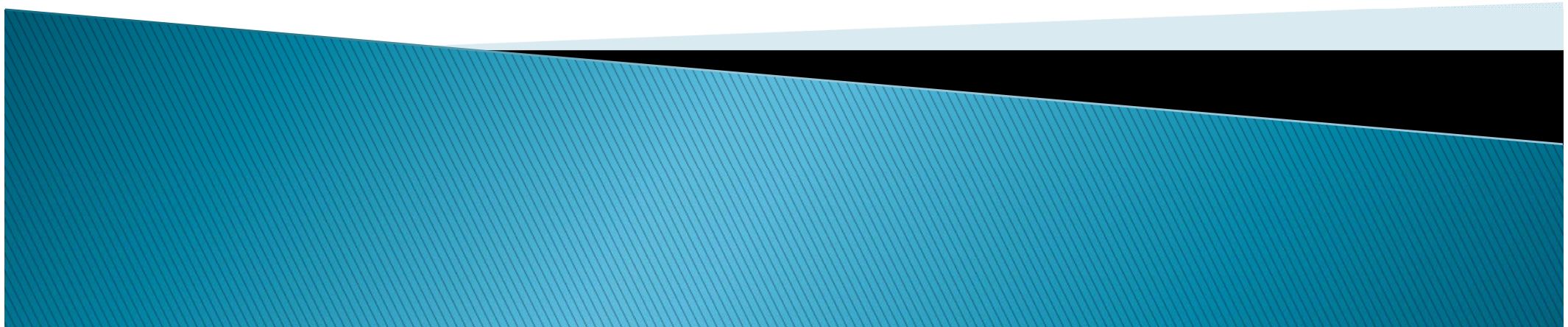
# Εισαγωγή στη βιβλιοθήκη CGAL

Δρ. Γιάννης Χαμόδρακας
Μέλος ΕΔΙΠ του Τμήματος Πληροφορικής και Τηλεπικοινωνιών
e-mail: ihamod@di.uoa.gr

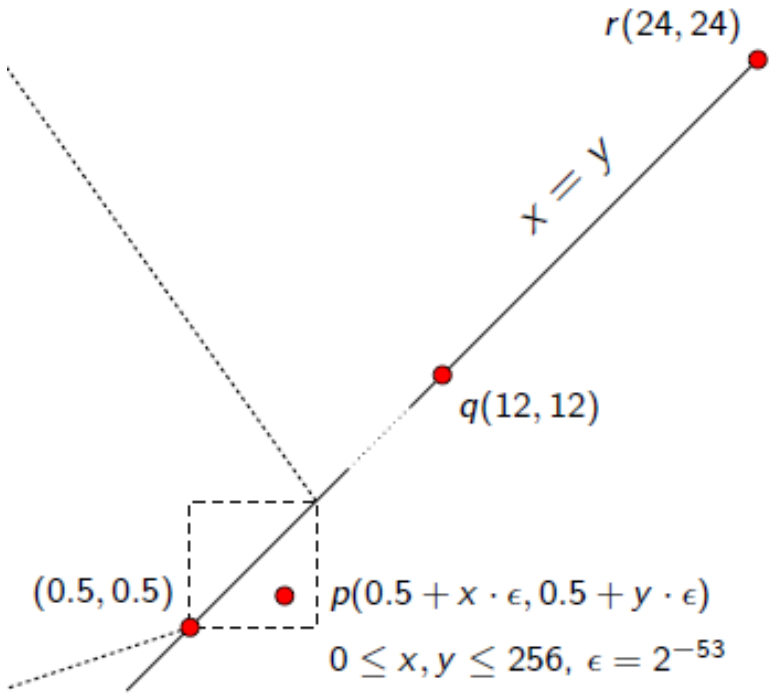


# Computational Geometry Algorithms Library

- Βιβλιοθήκη αναφοράς για αλγορίθμους Υπολογιστικής Γεωμετρίας στην C++
- Real RAM Model
  - Οι πράξεις μεταξύ πραγματικών αριθμών δίνουν ακριβή αποτελέσματα
  - O(1) το υπολογιστικό κόστος των βασικών πράξεων
  - Η CGAL αποτελεί προσέγγιση του μοντέλου μέσω του Exact Geometric Computing Paradigm
- Γενικευμένος αντικειμενοστραφής προγραμματισμός (Traits, Concepts, Models)

# Προβλήματα αριθμητικής floating point σε γεωμετρικά προβλήματα

- Τα σημεία πάνω στην ευθεία x=y, όταν x και y είναι συνήθεις floating point δεν είναι απαραίτητα συνευθειακά(!)



# Multiple precision floating point

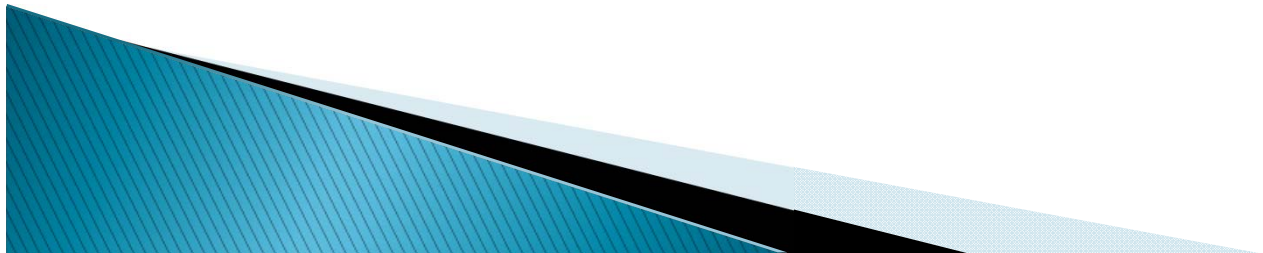
- Υπάρχουν υλοποιήσεις που υποστηρίζουν όλους τους πραγματικούς αριθμούς –εξαιρούνται οι υπερβατικοί (βιβλιοθήκες *CORE και LEDA)*
- Το υπολογιστικό κόστος είναι μεγάλο
- Η ακρίβεια περιορίζεται μόνο από το μέγεθος της μνήμης

# Exact Geometric Computing Paradigm

- Έλεγχος κατηγορημάτων με απλούς floating point αριθμούς
- Αν το αποτέλεσμα είναι αβέβαιο:
  - Χρήση multiple precision floating point αριθμών

# Γενικευμένος Προγραμματισμός

- Templates στην C++, Generics στην Java
- Στόχος: μοναδική υλοποίηση αλγορίθμων για πολλαπλούς τύπους
- Template κλάσεων: Πρότυπα για την παραγωγή κλάσεων βάσει παραμέτρων τύπου
- Template συναρτήσεων: Πρότυπα για την παραγωγή συναρτήσεων βάσει παραμέτρων τύπου
- Ρητές εξειδικεύσεις προτύπων: όταν η περιγραφή του προτύπου δεν είναι συμβατή με συγκεκριμένο τύπο



# Παραδείγματα Template

```
//Πρότυπο Στοίβας
template <typename T>
class Stack {
        public:...
        private:
        int size;
        int top;
        T *stackPtr;
};
template <typename T>
Stack<T>::Stack(int s)
        :size(s),
        top(-1),
        stackPtr(new T[size])
{}
```

```
//Ρητή εξειδίκευση για τύπο Point
template <>
class Stack<Point> { . . .};
template <>
Stack<Point>::Stack(int s)
        :size(s),
        top(-1),
        stackPtr(new Point[size])
{}
```

# Παραδείγματα Template (STL)

```
template <class InputIterator, class T>
InputIterator find(InputIterator first, InputIterator last, const T &value);
```

Χρήση κλάσης που παράγεται από template:

```
std::vector<int> v;
for(std::vector<int>::iterator it=v.begin();it!=v.end();++it){}
```

"***Input iterators*** can move only forward, can move only one step at a time, can only read what they point to, and can read what they're pointing to only once. They're modeled on the read pointer into an inputfile; the C++ library's istream_iterators are representative of this category.

***Output iterators*** are analogous, but for output: they move only forward, move only one
Step at a time, can only write what they point to, and can write it only once. They 're modeled on the write pointer into an output file; ostream_iterators epitomize this category…

A more powerful iterator category consists of ***forward iterators.*** Such iterators can do everything input and output iterators can do, plus they can read or write what they point to more than once.

***Bidirectional iterators*** add to forward iterators the ability to move backward as well as forward. Iterators for the STL's list are in this category.

The most powerful iterator category is that of ***random access iterators***. These kinds of iterators add to bidirectional iterators the ability to perform "iterator arithmetic," i.e., to jump forward or backward an arbitrary distance in constant time. Such arithmetic is analogous to pointer arithmetic, which is not surprising, because random access iterators are modeled on **built-in pointers**, and built-in pointers can act as random access iterators. Iterators for vector, deque, and string are random access iterators."(Scott Meyers, *Effective C++, 3rd edition*)

# Κατηγορίες προτύπων / τύπων STL

- **Containers**: Πρότυπα κλάσεων των οποίων «στιγμιότυπα» αποθηκεύουν συλλογές αντικειμένων συγκεκριμένου τύπου
- **Iterators:** Γενίκευση των δεικτών: αντικείμενα που «δείχνουν» σε άλλα αντικείμενα
- **Functors:** Κλάσεις για τις οποίες είναι ορισμένος ο τελεστής () και επομένως μπορούν να χρησιμοποιηθούν όπως οι συναρτήσεις – μέσω των πεδίων έχουν το πλεονέκτημα να αποθηκεύουν κατάσταση
- **Allocators:** Αντικείμενα που αποδίδουν μνήμη
- **Adaptors:** Τύποι που μετασχηματίζουν τη διεπαφή άλλων τύπων

# Παράδειγμα χρήσης vector και iterator
## [V.Fisikopoulos, Introduction to CGAL]

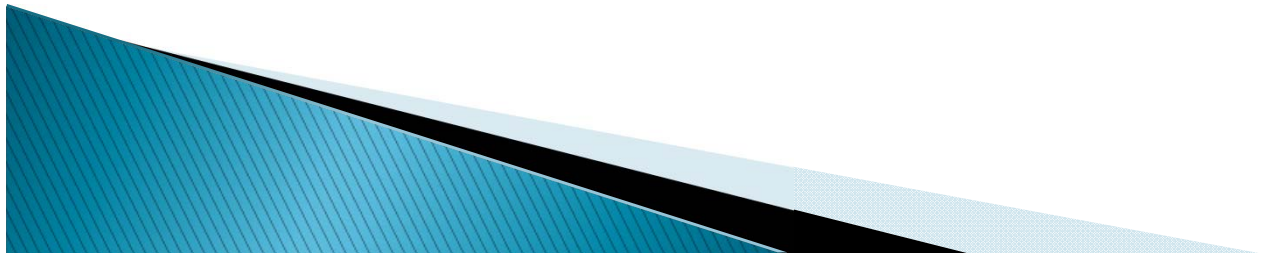
```
#include <iostream>
#include <vector>

typedef std::vector<double> dvector;
typedef std::vector<double>::iterator dveciterator;
int main() {

   //construct a vector
   dvector p;
   //insert elements into vector
   p.pushback(0);
   p.pushback(10);
   p.pushback(10);
   p.pushback(6);
   p.pushback(4);
   //iterate through vector's elements and print them
   for(dveciterator it=p.begin();it!=p.end();++it)
        std::cout << *it << std::endl;
   return 0; }
```

# Γλωσσάρι Γενικευμένου Προγραμματισμού στη CGAL

- Concept: Σύνολο απαιτήσεων που πρέπει να καλύπτει μία κλάση
- Model: Κλάση που συμμορφώνεται με ένα Concept
- Traits: Δομές που παρέχουν πληροφορίες για έναν τύπο κατά το χρόνο μεταγλώττισης / Models που περιγράφουν συμπεριφορά
- Refinement: επέκταση των απαιτήσεων ενός Concept
- Generalization: μείωση απαιτήσεων ενός Concept

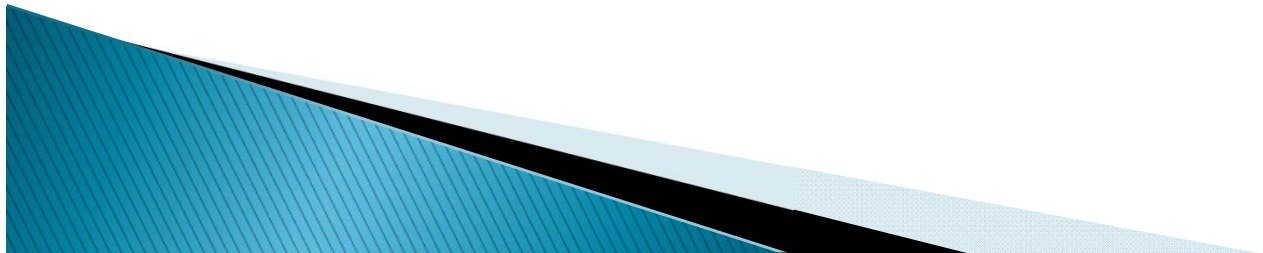


# Concepts και Models με ένα απλό παράδειγμα ([E.Fogel,TAU,IL,2012])

```
Template <typename T> void swap(T&a, T&b)
{T tmp(a); a=b; b=tmp;}
```

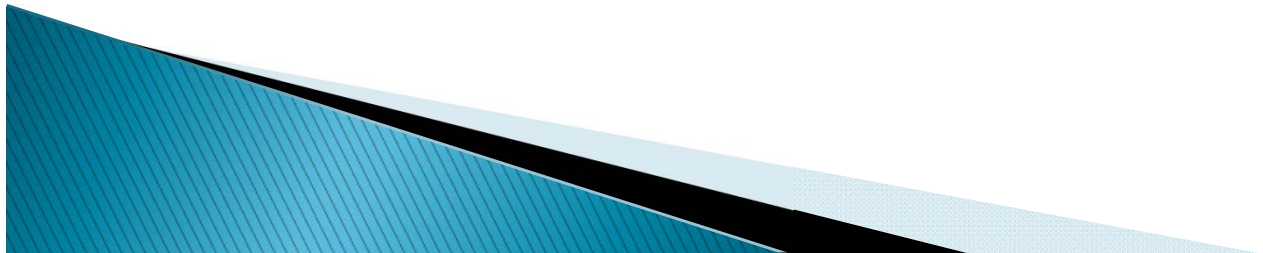
- Ο τύπος T πρέπει να διαθέτει:
  - Copy constructor
  - Copy assignment operator
- Ο τύπος T είναι Model του Concept CopyConstructible
- Ο τύπος T είναι Model του Concept Assignable
- Π.χ. Ο τύπος int είναι model και των 2 concepts

```
int a=2, b=4; std::swap(a,b);
```

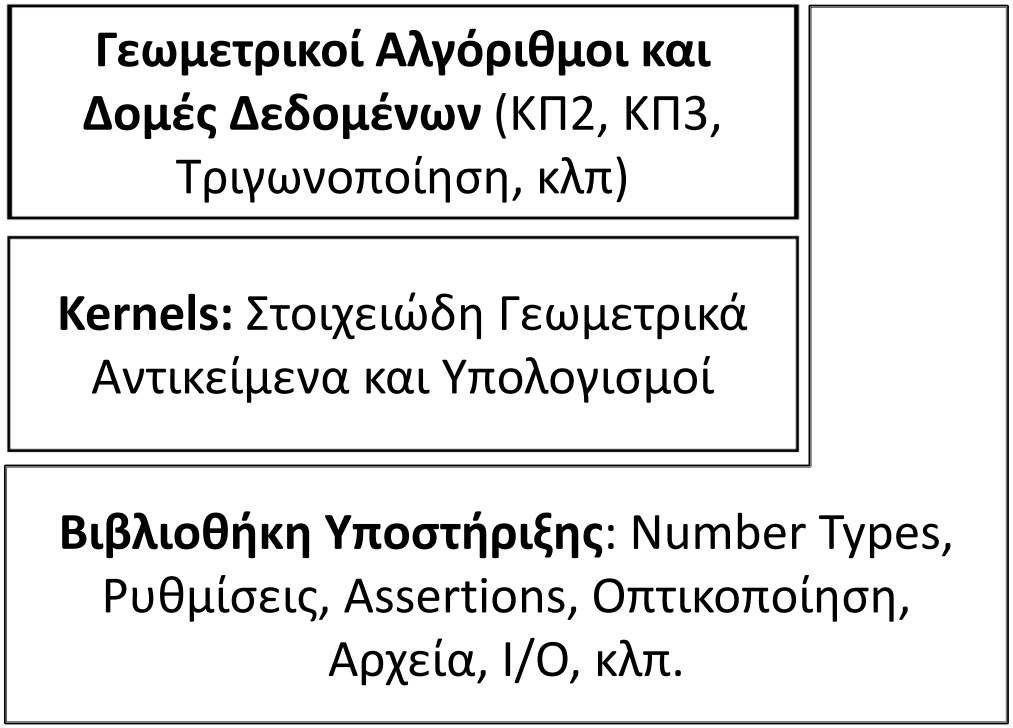


# Πληροφορίες για τη CGAL

- Υλοποίηση στη C++ από το 1995
- ~$10^6$ γραμμές κώδικα
- Χειρίζεται ρητά τις εκφυλισμένες γεωμετρικές περιπτώσεις (π.χ. όταν η διάσταση γεωμετρικών δεδομένων είναι μικρότερη από την αναμενόμενη)



# Αρχιτεκτονική της CGAL



# Kernels

- Στοιχείωδη γεωμετρικά αντικείμενα (2D, 3D, dD): Point, Vector, Triangle, Circle, κλπ.
- Στοιχειώδεις γεωμετρικές λειτουργίες (έλεγχος κατηγορημάτων (προσανατολισμού, σύγκρισης, κλπ), κατασκευές (εύρεση τομής, μέσου σημείου, κλπ).

  point - origin → vector

  point - point → vector

  point + vector → point

  point + point ← Illegal

- Αναπαράσταση συντεταγμένων: cartesian double, exact predicates inexact constructions, exact predicates exact constructions

# Kernels types & objects

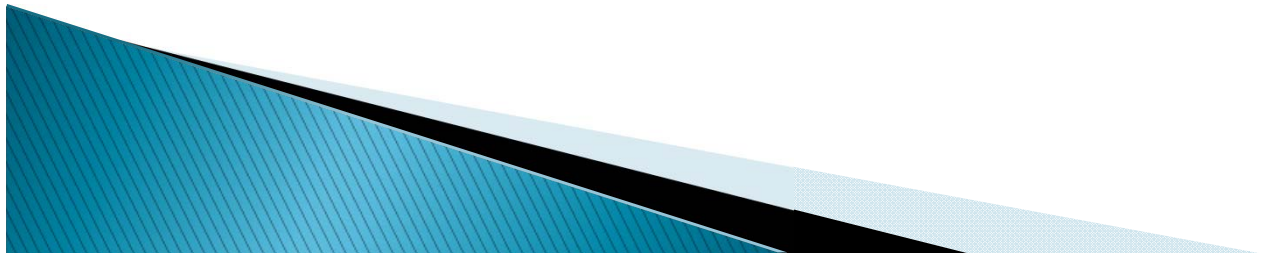
```
typedef CGAL::Cartesian<double> Kernel;
typedef CGAL::Exact_predicates_exact_constructions_kernel K2;

typedef Kernel::Point_2 Point_2;
typedef K2::Point_2 Point_2EX;

typedef std::vector<Point 2> Points;
```

- Το κατάλληλο Kernel επιλέγεται ανάλογα με το είδος του προβλήματος. Αν π.χ. γίνεται μόνο έλεγχος κατηγορημάτων καλύτερα να χρησιμοποιείται το:

```
typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
```



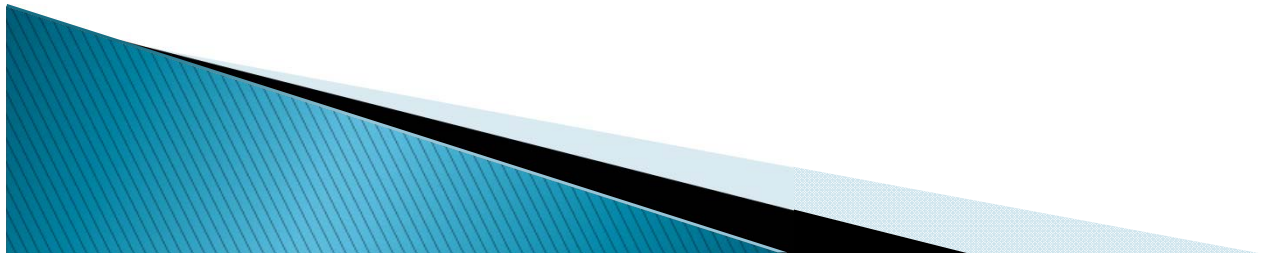
# Εγκατάσταση CGAL

- Σε συστήματα Ubuntu:
  - sudo apt-get install libcgal-dev (απαιτείται ο μεταγλωττιστής g++ και το cmake, examples: sudo apt-get install libcgal-demo)
  - $CGAL_HOME: /usr/lib/CGAL
- Σε άλλα συστήματα:
  - http://doc.cgal.org/latest/Manual/installation.html

# Μεταγλώττιση

```
cd /path/to/program
cgal_create_CMakeLists -s <desired_name_of_executable>
cmake -DCGAL_DIR=/usr/lib/CGAL .
make
```

(Αν έχει εγκατασταθεί με το apt-get σε Debian/Ubuntu, διαφορετικά το CGAL_DIR μπορεί να βρίσκεται αλλού)



# Ποια θα είναι η έξοδος του προγράμματος;

```
#include <iostream>
#include <CGAL/Simple_cartesian.h>
typedef CGAL::Simple_cartesian<double> Kernel;
typedef Kernel::Point_2 Point_2;
int main()
{
{
   Point_2 p(0, 0.3), q(1, 0.6), r(2, 0.9);
   std::cout << (CGAL::collinear(p,q,r) ? "collinear\n" : "not" "collinear\n");
}
{
   Point_2 p(0, 0.4), q(1, 0.8), r(2, 1.2);
   std::cout << (CGAL::collinear(p,q,r) ? "collinear\n" : "not" "collinear\n");
}
{
   Point_2 p(0,0), q(1, 1), r(2, 2);
   std::cout << (CGAL::collinear(p,q,r) ? "collinear\n" : "not" "collinear\n");
}
return 0;
}
```

# Έξοδος

non-collinear

non-collinear

Collinear

Γιατί; Προτείνετε διορθώσεις.

# Λύση

```
#include <iostream>
#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
#include <sstream>

typedef CGAL::Exact_predicates_exact_constructions_kernel Kernel;
typedef Kernel::Point_2 Point_2;
int main()
{
        Point_2 p, q, r;
        {
                std::istringstream input("0 0.3   1 0.6   2 0.9");
                input >> p >> q >> r;
                std::cout << (CGAL::collinear(p,q,r) ? "collinear\n" : "not"
                " collinear\n");
        }
        {
                std::istringstream input("0 0.4   1 0.8   2 1.2");
                input >> p >> q >> r;
                std::cout << (CGAL::collinear(p,q,r) ? "collinear\n" : "not"
                " collinear\n");
        }
        {
                Point_2 p(0,0), q(1, 1), r(2, 2);
                std::cout << (CGAL::collinear(p,q,r) ? "collinear\n" : "not"
                "collinear\n");
        }
        return 0;
}
```

# Παράδειγμα ΚΠ2 με ανάγνωση από αρχείο

```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/convex_hull_2.h>
#include <fstream>
typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef K::Point_2 Point_2;
typedef std::istream_iterator< Point_2 > point2_iterator;
typedef std::vector<Point_2> Points;
typedef std::vector<Point_2>::iterator pveciterator;

int main()
{
  Points result;
  std::ifstream in("data.in");
  point2_iterator begin(in);
  point2_iterator end;

  CGAL::convex_hull_2( begin, end, std::back_inserter(result) );
  std::cout << result.size() << " points on the convex hull" << std::endl;
  for (pveciterator iter=result.begin(); iter!=result.end(); ++iter)
        std::cout <<  *iter << std::endl;
  return 0;
}
```