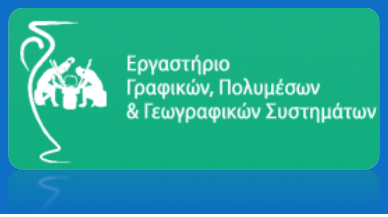
Standard Template Library (STL)

C++ library

Δομές Δεδομένων

Μάριος Κενδέα



10 Μαρτίου 2015

kendea@ceid.upatras.gr

Εισαγωγή

- Η Standard Βιβλιοθήκη προτύπων (STL) είναι μια βιβλιοθήκη λογισμικού η οποία περιλαμβάνεται στην C++ Standard Library.
- Παρέχει:
  - Containers
  - Iterators
  - Algorithms
  - Functors

Πλεονεκτήματα της Standard Template Library (STL) (1/2)

- Παρέχει δυνατότητα χρήσης γενικών δομών δεδομένων και αλγορίθμων.
- Αποτελείται από ένα ισχυρό και ευέλικτο σύνολο κλάσεων και λειτουργιών.
- Παρέχει ένα αποτελεσματικό, ελαφρύ, και επεκτάσιμο πλαίσιο για την ανάπτυξη εφαρμογών πληροφορικής.
- Προσφέρει ένα εκλεπτυσμένο επίπεδο αφαίρεσης που προωθεί τη χρήση των γενικών δομών δεδομένων και αλγορίθμων χωρίς την επιβάρυνση μιας γενικής λύσης.
- Είναι αρκετά ευέλικτη στην αντιμετώπιση των αναπτυξιακών αναγκών της για κάθε είδους εφαρμογές.

Πλεονεκτήματα της Standard Template Library (STL) (2/2)

- Οι εφαρμογές δεν υπόκεινται σε μείωση της απόδοσης τους μέσω της χρήσης της STL, δεδομένου ότι η STL παρέχει συγκεκριμένες εγγυήσεις απόδοσης για τους αλγόριθμους που προμηθεύει και μπορεί να επιλεγεί αυτός που ανταποκρίνεται καλύτερα στις ανάγκες της κάθε εφαρμογής
- Απαιτεί μικρότερο χρόνο ανάπτυξης και debugging
- Επιτυγχάνει καλή διαχείριση μνήμης
- Απαιτείται μικρό μέγεθος κώδικα.
- Επιτυγχάνει τα αποτελέσματά της μέσω της χρήσης των προτύπων templates.
- Η προσέγγιση χρήσης των προτύπων παρέχει πολυμορφισμό χρόνου μεταγλώττισης που είναι συχνά πιο αποδοτικός από ό, τι ο παραδοσιακός πολυμορφισμός χρόνου εκτέλεσης

Βασικά συστατικά της Standard Template Library (STL)

- Η STL παρέχει ένα σύνολο κλάσεων C++, όπως οι **Sequence Containers, οι Associative Containers και οι Container Adaptors**
- Ένας **container** είναι μια κλάση, μια δομή δεδομένων, ή ένας αφηρημένος τύπος στοιχείων (Abstract Data Type) του οποίου τα στιγμιότυπα είναι συλλογές άλλων αντικειμένων. Με άλλα λόγια χρησιμοποιούνται για να αποθηκεύσουν τα αντικείμενα με οργανωμένο τρόπο ακολουθώντας συγκεκριμένους κανόνες πρόσβασης.

Sequence Containers (1/8)

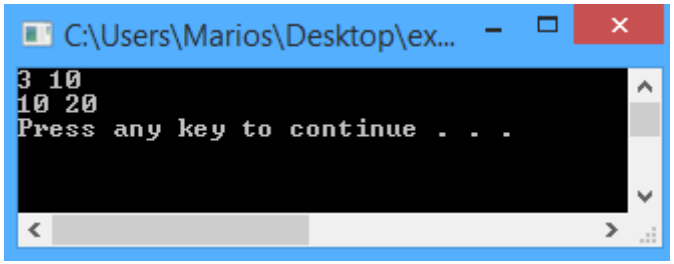
- Διατεταγμένοι πίνακες αντικειμένων μεταβλητού μήκους
  - vector
  - deque
  - list
- Επιτρέπουν εισαγωγή και διαγραφή στοιχείων σε οποιοδήποτε σημείο.
- **Βασική διαφορά:** Το κόστος εξαρτάται από τον τύπο.

Sequence Containers (2/8)
Vector (1/2)

- Είναι ένας δυναμικός πίνακας.
- Επιτρέπει τυχαία προσπέλαση, O(1)
- Υπάρχει δυνατότητα να επανα-ταξινομηθεί αυτόματα, κατά την προσθήκη ή τη διαγραφή ενός αντικειμένου.
- Το κόστος εισαγωγής στο τέλος ενός διανύσματος είναι σταθερό, O(1).
- Το κόστος εισαγωγής στην αρχή ενός διανύσματος είναι γραμμικό, O(n).
- Το κόστος εισαγωγής στη μέση ενός διανύσματος είναι γραμμικό, O(n).

Sequence Containers (3/8) Vector (2/2)

```
1 #include <vector>
2
3 int main(){
4   vector <int> example;
5   example.push_back(3);
6   example.push_back(10);
7
8   for(int x=0; x<example.size(); x++){
9       cout << example[x] << " ";
10  }
11  cout << endl;
12  if (!example.empty())
13      example.clear();
14
15  vector<int> another_vector;
16  another_vector.push_back(10);
17  example.push_back(10);
18  if (example == another_vector)
19      example.push_back(20);
20
21  for (int y=0; y<example.size(); y++){
22      cout << example[y] << " ";
23  }
24  cout << endl;
25  return 0;
26}
```



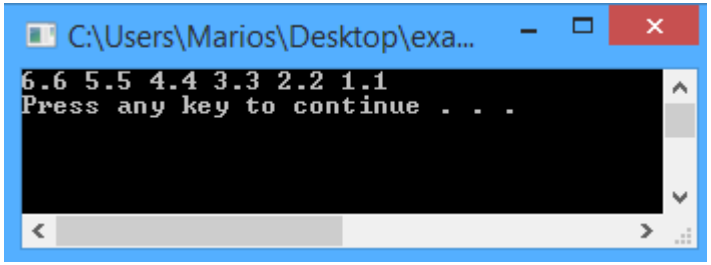
Sequence Containers (4/8)
Deque (1/3)

- Το κόστος εισαγωγής/διαγραφής στην αρχή είναι σταθερό, O(1).
- Το κόστος εισαγωγής/διαγραφής στο τέλος είναι σταθερό, O(1).
- Το κόστος εισαγωγής/διαγραφής στη μέση είναι γραμμικό, O(n).
- Επιτρέπει τυχαία προσπέλαση, O(1).
- Δεν παρέχει εγγυήσεις εγκυρότητας όταν έχει αλλάξει η σειρά.

Sequence Containers (5/8) Deque (2/3)

```
/*
 * Constructors
*/
    deque<int> first; //empty

    deque<int> second (4, 100); // 4 ints = 100

    deque<int> third(second.begin(), second.end()); //iterating through second

    deque<int> fourth(third); //a copy of third

    deque<double> fifth(5, 8.1); //doubles with 5 elements set to 8.1
```

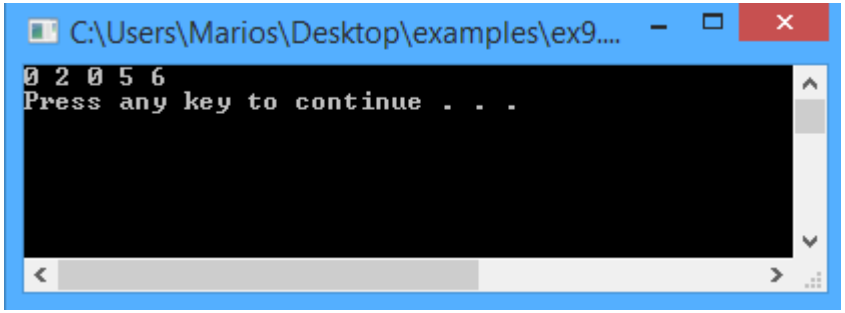
```
1 #include <deque>
2
3 int main(){
4
5   deque<float> example;
6   for(int i=1; i<=6; ++i){
7       example.push_front(i*1.1);
8   }
9
10  for(int i=0; i<example.size(); ++i){
11      cout << example[i] << " ";
12  }
13  cout << endl;
14  return 0;
15}
```



Sequence Containers (7/8)
List (1/2)

- Σε μία διπλά-συνδεμένη λίστα, τα στοιχεία δεν αποθηκεύονται στην παρακείμενη μνήμη.
- Επιτυγχάνει αντίθετη απόδοση από ένα διάνυσμα.
- Αργή αναζήτηση και προσπέλαση (γραμμικός χρόνος)…
- …αλλά μόλις βρεθεί μια θέση, γρήγορη εισαγωγή και διαγραφή (σταθερός χρόνος).
- Το κόστος εισαγωγής στην αρχή, στο τέλος, στη μέση είναι σταθερό.
- Επιτρέπει όμως **μόνο** σειριακή προσπέλαση.

```
1 #include <list>
2
3 int main()
4 {
5    list<int> L;
6    L.push_back(0);
7    L.push_front(0);
8    L.insert(++L.begin(),2);
9
10   L.push_back(5);
11   L.push_back(6);
12
13   list<int>::iterator i;
14
15   for(i=L.begin(); i != L.end(); ++i)
16      cout << *i << " ";
17   cout << endl;
18   system ("pause");
19   return 0;
20}
```



Associative Containers (1/5)

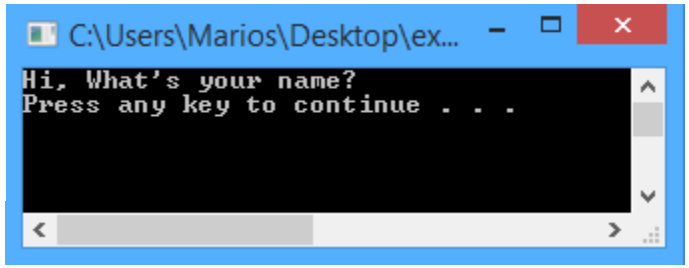
- Γενίκευση των sequences
  - map - multimap
  - set - multiset
- **Βασική διαφορά:** Δεν δεικτοδοτούνται από ακεραίους αλλά από κάποιο κλειδί.
- Το κλειδί αυτό μπορεί να είναι οποιουδήποτε τύπου

Associative Containers (2/5)
Map – MultiMap (1/2)

- Αποθηκεύει ζεύγη της μορφής <key, value>.
- Κάτι σαν πίνακας κατακερματισμού (Hash table), αλλά επιτρέπει εισαγωγή, διαγραφή και αναζήτηση με κόστος λογαριθμικό.
- Επιτρέπει ακόμα σειριακή προσπέλαση όλων των στοιχείων.
- Δεν επιτρέπονται διπλότυπα κλειδιά.
- MultiMap: Όμοιο με το map με τη διαφορά ότι επιτρέπονται διπλότυπα κλειδιά

hash set - hash multiset - hash map - hash multimap. Παρόμοια με τα set, multiset, map, multimap, αλλά υλοποιούνται με ένα hashtable. Τα κλειδιά δεν διατάσονται, αλλά μια hash λειτουργία είναι απαραίτητη για το βασικό τύπο. Αυτοί οι containers δεν είναι μέρος της C++ Standard Library, αλλά συμπεριλαμβάνονται στις επεκτάσεις STL SGI, και συμπεριλαμβάνονται σε βιβλιοθήκες όπως η GNU C++.

```
1 #include <map>
2
3 int main(){
4   typedef multimap<int, string> IntStringMMap;
5   IntStringMMap example;
6
7   example.insert(make_pair(2, "your"));
8   example.insert(make_pair(1, "Hi,"));
9   example.insert(make_pair(1, "What's"));
10  example.insert(make_pair(3, "name?"));
11
12  IntStringMMap::iterator pos ;
13  for(pos=example.begin(); pos!=example.end(); ++pos) {
14  cout << pos->second << " " ;
15  }
16  cout << endl;
17}
```

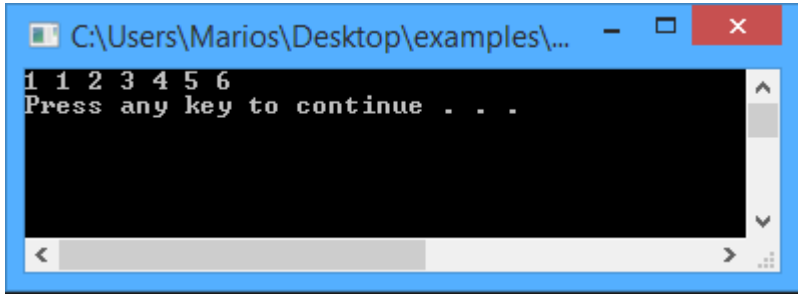


Associative Containers (4/5)
Set – MultiSet (1/2)

- Η εισαγωγή/διαγραφή των στοιχείων σε ένα σύνολο δεν ακυρώνει τους iterators που δείχνουν στο σύνολο.
- Παρέχει τις λειτουργίες: ένωση, τομή, διαφορά, συμμετρική διαφορά.
- Όμοιο με το map με τη διαφορά ότι οι ίδιες οι τιμές παίζουν και το ρόλο του κλειδιού.
- Επιτρέπει εισαγωγή, διαγραφή και αναζήτηση με κόστος λογαριθμικό.
- Επιτρέπει σειριακή προσπέλαση όλων των στοιχείων.
- Δεν επιτρέπονται διπλότυπα.
- Τα δεδομένα πρέπει να χρησιμοποιούν τον τελεστή σύγκρισης <.
- Έχει υλοποιηθεί χρησιμοποιώντας δυαδικό δέντρο αναζήτησης εξισορρόπησης.
- Multiset: Όμοιο με το set με τη διαφορά ότι επιτρέπονται διπλότυπα.

Associative Containers (5/5) Set - MultiSet (2/2)

```
1  #include <set>
2
3  int main() {
4    typedef std::multiset<int> IntSet;
5    IntSet myset;
6
7    myset.insert(3);
8    myset.insert(1);
9    myset.insert(5);
10   myset.insert(4);
11   myset.insert(1);
12   myset.insert(6);
13   myset.insert(2);
14
15       IntSet::const_iterator pos;
16     for (pos = myset.begin(); pos != myset.end(); ++pos) {
17         cout << *pos << ' ';
18     }
19     cout << endl;
20 }
```



Container Adaptors (1/2)

- **Ορισμός:** Η ουρά (queue), ουρά προτεραιότητας (priority queue) και η στοίβα (stack) αποτελούν Container Adaptors με συγκεκριμένη διεπαφή, που χρησιμοποιούν άλλους containers ως εφαρμογή.

- **Ουρά:**
  - Παρέχει μία διεπαφή ουράς FIFO για τις λειτουργίες push,pop,front,back.
  - Οποιεσδήποτε λειτουργίες υποστήριξης σειρών όπως η front(), back(), push_back(), pop_front() μπορεί να χρησιμοποιηθεί ως στιγμιότυπο της ουράς (π.χ. list, deque).

Container Adaptors (2/2)

- **Ουρά Προτεραιότητας:**
  - Παρέχει μία διεπαφή ουράς LIFO για τις λειτουργίες push, pop, top. (το στοιχείο με την υψηλότερη προτεραιότητα είναι στην κορυφή).
  - Οποιαδήποτε σειρά τυχαίας προσπέλασης που υποστηρίζει τις λειτουργίες front(), push_back(), pop_back() μπορεί να χρησιμοποιηθεί ως στιγμιότυπο της ουράς προτεραιότητας (π.χ. vector, deque).
  - Τα στοιχεία πρέπει, επίσης, να υποστηρίζουν τη σύγκριση (για να καθοριστεί ποιο στοιχείο έχει υψηλότερη προτεραιότητα και πρέπει να βγαίνει πρώτο από την ουρά).
- **Στοίβα:**
  - Παρέχει μία διεπαφή ουράς LIFO για τις λειτουργίες push,pop,top. (το τελευταίος-στοιχείο είναι στην κορυφή).
  - Οποιεσδήποτε λειτουργίες υποστήριξης σειρών όπως η back(), push_back(), pop_back() μπορεί να χρησιμοποιηθούν ως στιγμιότυπο της στοίβας (π.χ. vector, list, deque).

Iterators (1/4)

- Οι containers προσφέρουν δυνατότητα προσπέλασης των στοιχείων τους μέσω iterators.
- Πρόκειται για δείκτες στα στοιχεία ενός container.
- Βοηθούν στη σάρωση όλων των στοιχείων ενός container.
- Κάθε container ορίζει έναν ή περισσότερους τύπους iterators που μπορούμε να χρησιμοποιήσουμε.

Iterators (2/4)
Πέντε διαφορετικοί τύποι iterators

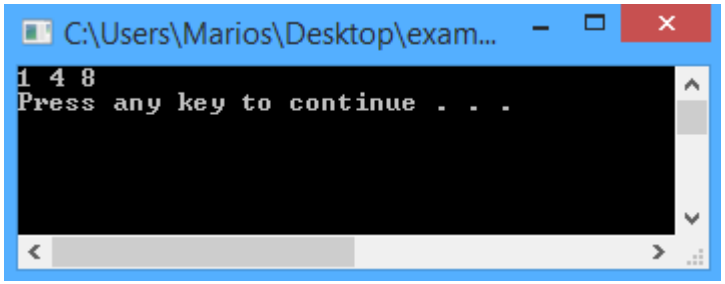
- input iterators (που μπορούν να χρησιμοποιηθούν μόνο για να διαβάσουν μια ακολουθία τιμών) *value = *myItr*
- output iterators (που μπορούν να χρησιμοποιηθούν μόνο για να γράψουν μια ακολουθία τιμών) **myItr = value*
- forward iterators (που μπορούν να διαβαστούν, να γράψουν, και να μετακινηθούν προς τα εμπρός) *++myItr is legal, but --myItr is not*
- bidirectional iterators (λειτουργούν όπως οι forward iterators, αλλά μπορούν επίσης να κινηθούν προς τα πίσω: list, set, multiset, map, and multimap)
- random access iterators (που μπορούν να κάνουν ελεύθερα οποιοδήποτε αριθμό βημάτων σε μια μόνο λειτουργία, vector and deque)

Iterators (3/4)
Μειονεκτήματα Χρήσης iterators

- Η χρήση των iterators επιφέρει κάποιο κόστος πολλές φορές.
- Π.χ. η εκτέλεση μιας αναζήτησης σε έναν associative container όπως σε ένα map ή σε ένα set μπορεί να είναι πολύ πιο αργή χρησιμοποιώντας τους iterators απ’ ό,τι θα ήταν αν χρησιμοποιούσαμε τις συναρτήσεις μέλη που προσφέρονται από τον ίδιο τον container.
- Αυτό συμβαίνει επειδή οι μέθοδοι ενός associative container μπορούν να εκμεταλλευτούν τη γνώση της εσωτερικής δομής, η οποία είναι αδιαφανής στους αλγορίθμους που χρησιμοποιούν τους iterators.

Iterators (4/4)

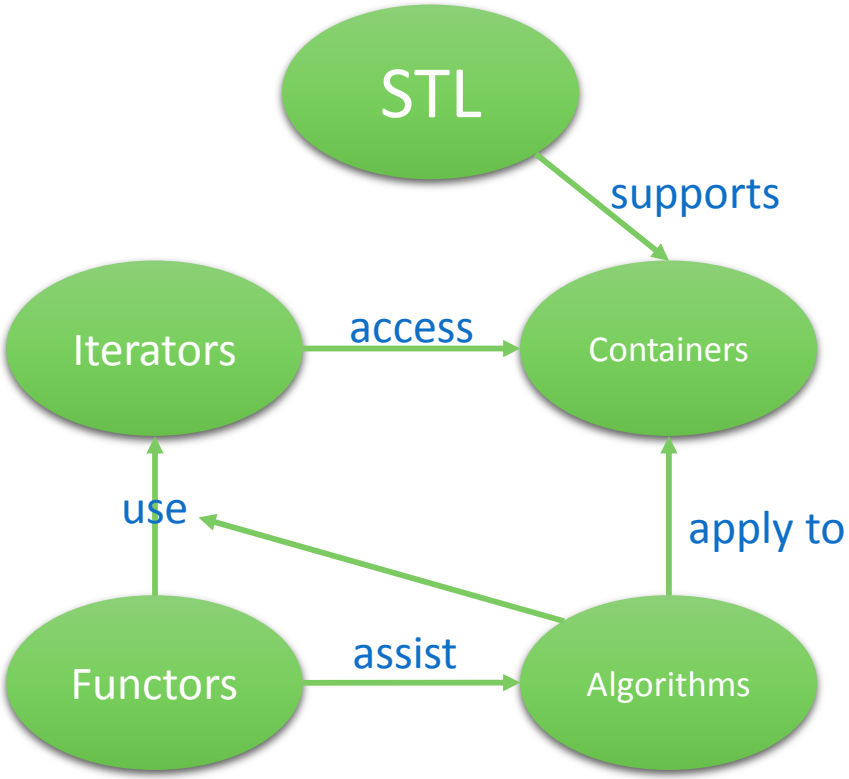
```
1 #include <vector>
2
3 int main(){
4   vector<int> myIntVector;
5   vector<int>::iterator myIntVectorIterator;
6
7   myIntVector.push_back(1);
8   myIntVector.push_back(4);
9   myIntVector.push_back(8);
10
11  for(myIntVectorIterator = myIntVector.begin(); myIntVectorIterator!=myIntVector.end(); myIntVectorIterator++) {
12      cout << *myIntVectorIterator << " " ;
13  }
14  cout << endl;
15}
```



Algorithms

- Πλούσια συλλογή από template C++ functions που λειτουργούν πάνω σε containers.
- Περιλαμβάνουν:
  - Ταξινόμηση (sort, merge, min, max,..)
  - Αναζήτηση (find, count, equal,..)
  - Μετασχηματισμούς (transform, replace, fill, rotate, nth element, random shuffle,..)
  - Πράξεις συνόλων (includes, set difference , set intersection, set union,..)

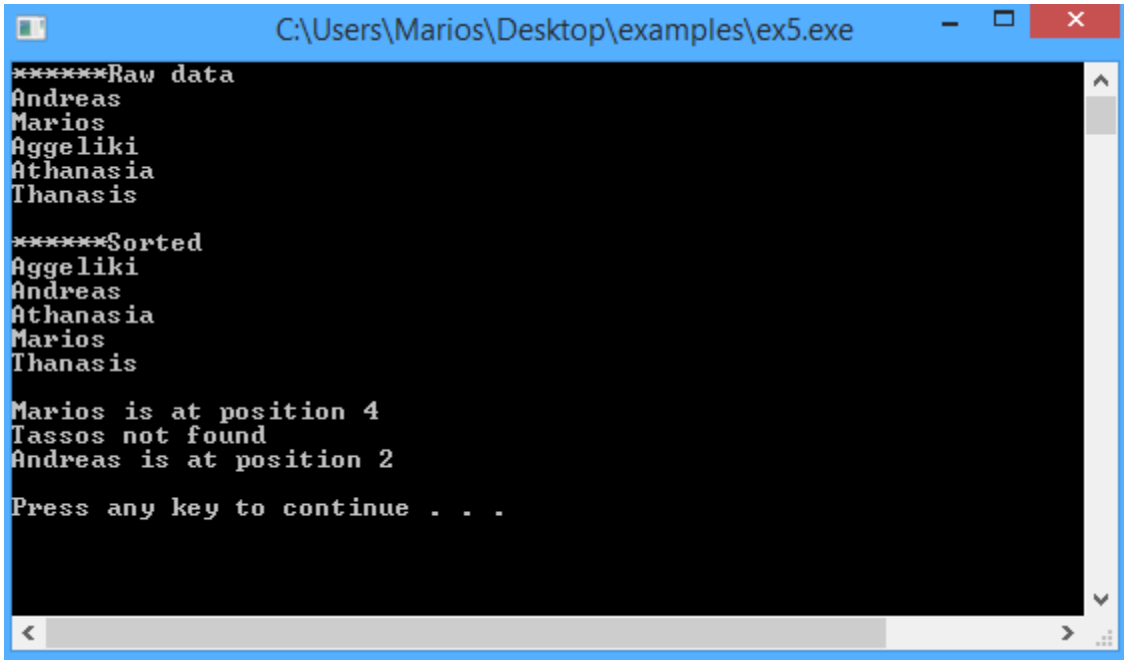
Ανακεφαλαίωση



Παράδειγμα (1)

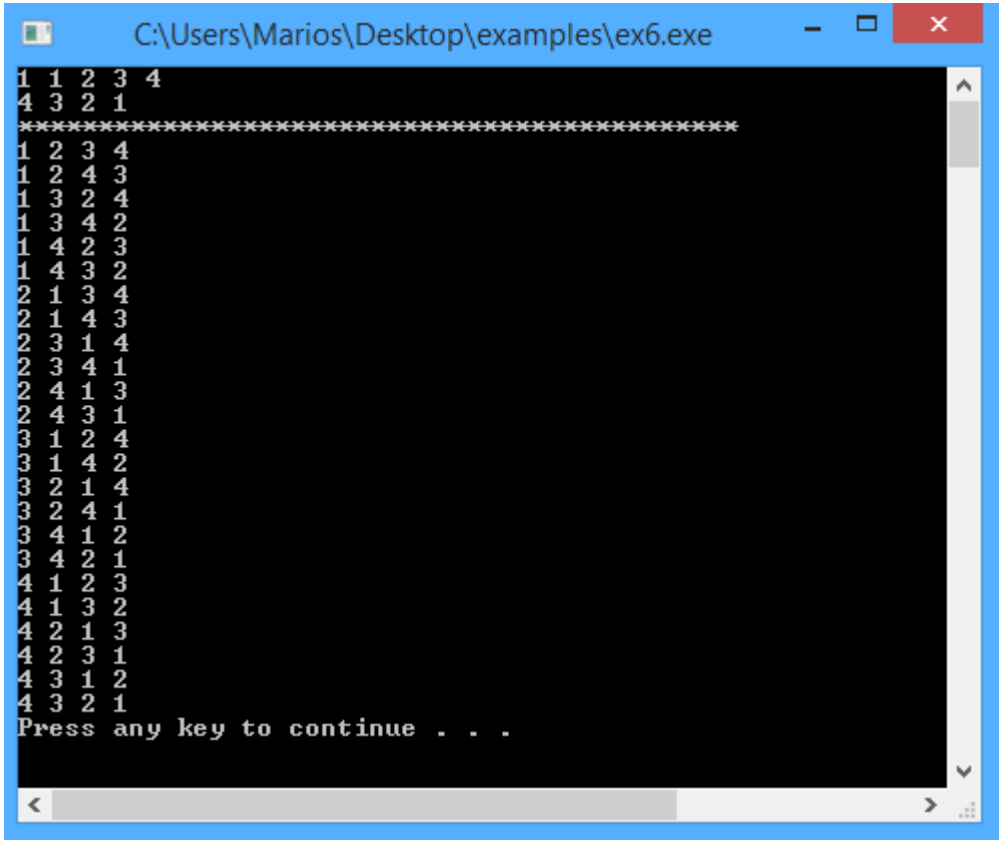
```
1 #include <iostream>
2 #include <vector>
3 #include <map>
4 #include <string>
5 #include <algorithm>
6
7 using namespace std;
8
9 void printAll(const vector<string> &sv){
10  for(int i=0; i<sv.size(); i++){
11      cout << sv[i].c_str() << endl;
12  }
13}
14
15 void fillMap(map<string, int> &theMap, const vector<string> &sv){
16  int i=0;
17  for(vector<string>::const_iterator it=sv.begin(); it!=sv.end(); it++){
18      theMap.insert(pair<string, int>(*it, i));
19      i++;
20  }
21}
22
23 void printPosition(const map<string, int> &theMap, string name){
24  map<string, int>::const_iterator it=theMap.find(name);
25  if(it==theMap.end())
26      cerr << name.c_str() << " not found" <<endl;
27  else
28      cout << name.c_str() << " is at position " << it->second +1 << endl;
29}
30
```

```
31 int main(int argc, char* argv[]){
32  vector<string> stringVector;
33
34  stringVector.push_back("Andreas");
35  stringVector.push_back("Marios");
36  stringVector.push_back("Aggeliki");
37  stringVector.push_back("Athanasia");
38  stringVector.push_back("Thanasis");
39
40  cout << "******Raw data" << endl;
41  printAll(stringVector);
42  cout << endl;
43  cout << "******Sorted" << endl;
44  sort(stringVector.begin(), stringVector.end());
45  printAll(stringVector);
46  cout << endl;
47  map<string, int> positionMap;
48  fillMap(positionMap, stringVector);
49  printPosition(positionMap, "Marios");
50  printPosition(positionMap, "Tassos");
51  printPosition(positionMap, "Andreas");
52
53  cout << endl;
54  system("pause");
55  return 0;
56}
```


C:\Users\Marios\Desktop\examples\ex5.exe
******Raw data
Andreas
Marios
Aggeliki
Athanasia
Thanasis
******Sorted
Aggeliki
Andreas
Athanasia
Marios
Thanasis
Marios is at position 4
Tassos not found
Andreas is at position 2
Press any key to continue . . .

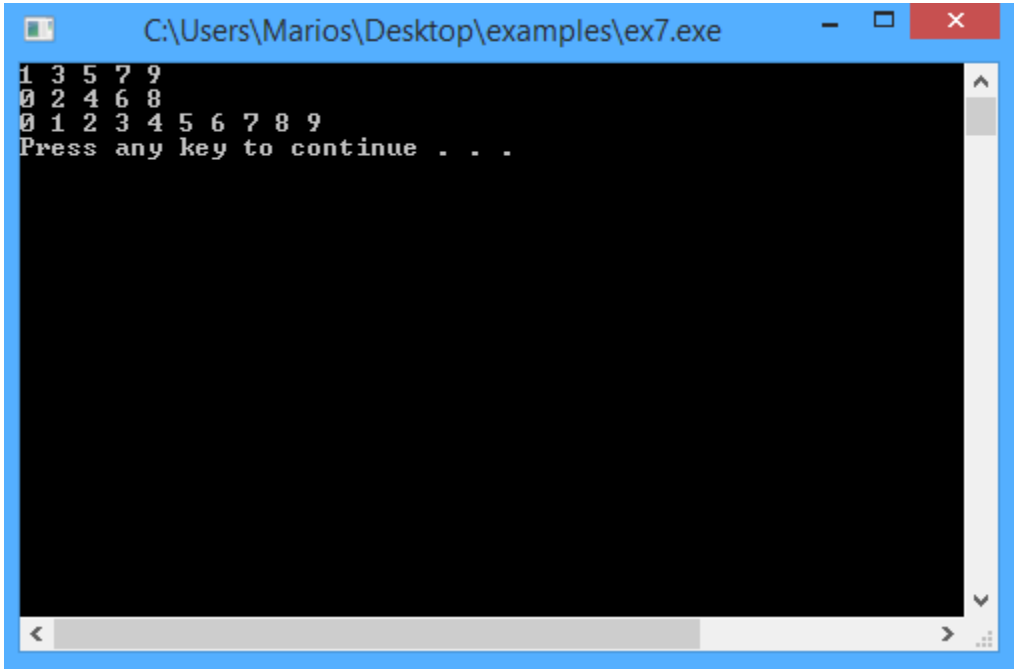
Παράδειγμα (2)

```
1 #include <iostream>
2 #include <list>
3 #include <algorithm>
4
5 using namespace std;
6
7 int main(int argc, char* argv[]){
8   list<int> numberList;
9
10  numberList.push_back(3);
11  numberList.push_back(4);
12  numberList.push_front(2);
13  numberList.push_front(1);
14  numberList.push_front(1);
15
16  for (list<int>::iterator it=numberList.begin(); it!=numberList.end(); it++){
17      cout << *it << " ";
18  }
19  cout << endl;
20
21  numberList.unique();
22  for (list<int>::reverse_iterator rit=numberList.rbegin(); rit!=numberList.rend(); rit++){
23      cout << *rit << " ";
24  }
25  cout << endl << "*****************************************" <<endl;
26
27  do{
28      for (list<int>::iterator it = numberList.begin(); it!=numberList.end(); it++){
29      cout << *it << " ";
30  }
31  cout << endl;
32  }while(next_permutation(numberList.begin(), numberList.end()));
33  system("pause");
34  return 0;
35}
```


C:\Users\Marios\Desktop\examples\ex6.exe
1 1 2 3 4
4 3 2 1
****
1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 2 3
1 4 3 2
2 1 3 4
2 1 4 3
2 3 1 4
2 3 4 1
2 4 1 3
2 4 3 1
3 1 2 4
3 1 4 2
3 2 1 4
3 2 4 1
3 4 1 2
3 4 2 1
4 1 2 3
4 1 3 2
4 2 1 3
4 2 3 1
4 3 1 2
4 3 2 1
Press any key to continue . . .

Παράδειγμα (3)

```
1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4
5 using namespace std;
6
7 void printAll(vector<int>::iterator begin, vector<int>::iterator end){
8   for (vector<int>::iterator it=begin; it!=end; it++)
9       cout <<*it <<" ";
10  cout <<endl;
11}
12
13 int main(int argc, char* argv[]){
14  vector<int> set1;
15  vector<int> set2;
16  vector<int> set3(10);
17
18  vector<int>::iterator startit=set3.begin();
19  vector<int>::iterator endit;
20
21  for (int i=0; i<10; i++)
22      set1.push_back(i);
23
24  for (int i=0; i<10; i+=2)
25      set2.push_back(i);
26
27  endit = set_difference(set1.begin(), set1.end(), set2.begin(), set2.end(), startit);
28  printAll(startit, endit);
29  endit = set_intersection(set1.begin(), set1.end(), set2.begin(), set2.end(), startit);
30  printAll(startit, endit);
31  endit = set_union(set1.begin(), set1.end(), set2.begin(), set2.end(), startit);
32  printAll(startit, endit);
33
34  system("pause");
35  return 0;
36}
```

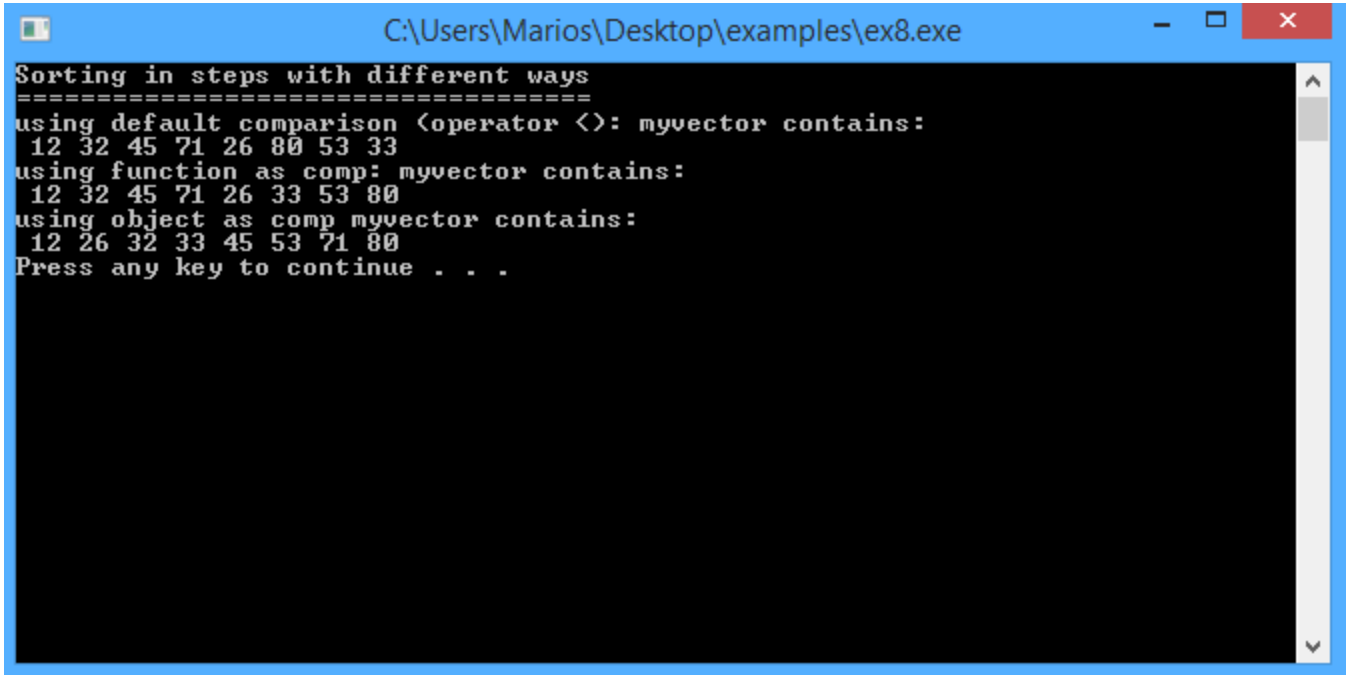

C:\Users\Marios\Desktop\examples\ex7.exe
1 3 5 7 9
0 2 4 6 8
0 1 2 3 4 5 6 7 8 9
Press any key to continue . . .

Παράδειγμα (4)

```
1  #include <iostream>
2  #include <algorithm>
3  #include <vector>
4
5  using namespace std;
6
7  bool myfunction (int i, int j){
8    return (i<j);
9    }
10
11 struct myclass {
12  bool operator() (int i, int j){
13       return (i<j);
14       }
15}      myobject;
16
```

```
17 int main () {
18  int myints[] = {32,71,12,45,26,80,53,33};
19  vector<int> myvector (myints, myints+8);
20
21  cout << "Sorting in steps with different ways" << endl;
22  cout << "====================================" << endl;
23  sort(myvector.begin(), myvector.begin()+4);
24  cout << "using default comparison (operator <): myvector contains:" << endl;
25  for (vector<int>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
26      cout << " " << *it;
27  cout << endl;
28
29  sort(myvector.begin()+4, myvector.end(), myfunction);
30  cout << "using function as comp: myvector contains:" << endl;
31  for (vector<int>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
32      cout << " " << *it;
33  cout << endl;
34
35  sort(myvector.begin(), myvector.end(), myobject);
36  cout << "using object as comp myvector contains:" << endl;
37  for (vector<int>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
38      cout << " " << *it;
39  cout << endl;
40
41  system ("pause");
42  return 0;
43}
```

Functor (Function Object) Χρήση operator() ως συνάρτηση


C:\Users\Marios\Desktop\examples\ex8.exe
Sorting in steps with different ways
==========
using default comparison (operator <): myvector contains:
12 32 45 71 26 80 53 33
using function as comp: myvector contains:
12 32 45 71 26 33 53 80
using object as comp myvector contains:
12 26 32 33 45 53 71 80
Press any key to continue . . .

References

- http://www.cplusplus.com/reference/stl/
- http://www.cprogramming.com/tutorial/stl/stlintro.html
- http://www.yolinux.com/TUTORIALS/LinuxTutorialC++STL.html
- https://www.sgi.com/tech/stl/
- http://www.stanford.edu/class/cs106l/course-reader/Ch7_STLAlgorithms.pdf

?