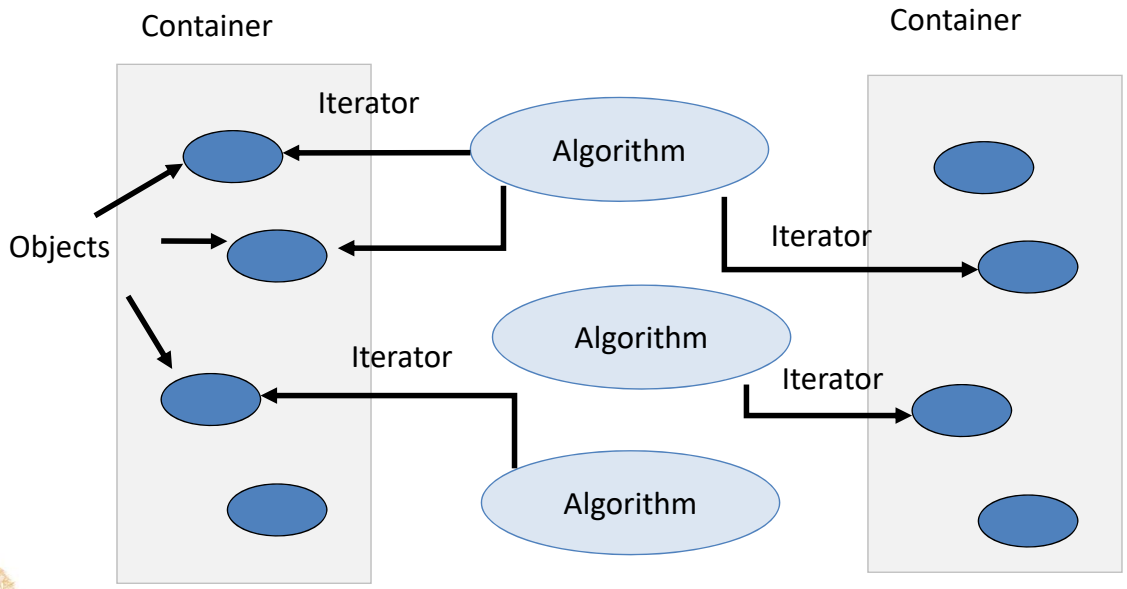
Βιβλιοθήκη STL

Standard Template Library

- Η standard template library (STL) αποτελείται από:
  - Containers
  - Algorithms
  - Iterators
- Ένας ***container*** είναι ένας τρόπος οργάνωσης μίας συλλογής αντικειμένων στην μνήμη
- ***Algorithms*** στην STL είναι διαδικασίες που εφαρμόζονται στους containers για να επεξεργαστούν τα δεδομένα τους, πχ. Αναζήτηση αντικειμένου
- ***Iterators*** είναι μία γενίκευση των δεικτών (pointers), δείχνουν αντικείμενα μέσα σε έναν container

Containers, Iterators, Algorithms

Οι αλγόριθμοι χρησιμοποιούν iterators για να αλληλεπιδράσουν με αντικείμενα των containers



Containers

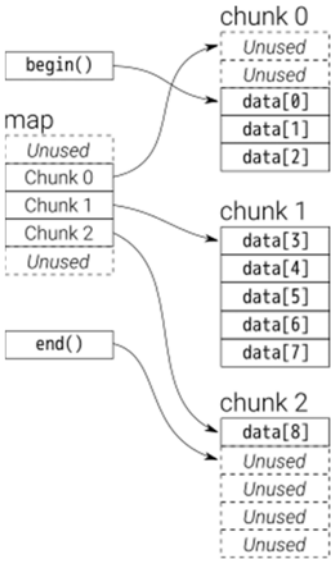
- Ένας **container** είναι ένας τρόπος να αποθηκεύσουμε δεδομένα, είτε βασικούς τύπους είτε αντικείμενα κλάσεων.
- Η STL παρέχει διάφορους βασικούς τύπους containers
  - <vector> : one-dimensional array
  - <list> : double linked list
  - <deque> : double-ended queue
  - <queue> : queue
  - <stack> : stack
  - <set> : set
  - <map> : associative array

Sequence Containers

- Ένας **sequence container** αποθηκεύει ένα σύνολο αντικειμένων σε ακολουθία, με άλλα λόγια κάθε αντικείμενο (εκτός από το πρώτο και το τελευταίο) ακολουθείται από ένα συγκεκριμένο αντικείμενο: <vector>, <list> and <deque>
  - **C++11:** <array>, <forward_list>
- Οι κλασικοί C++ πίνακες με σταθερό μέγεθος δεν μεταβάλλονται κατά την εκτέλεση. Έχουν το πλεονέκτημα της τυχαίας προσπέλασης
- <vector> είναι επεκτάσιμος τύπος αλλά οι εισαγωγές/διαγραφές στο μέσο είναι αργές.

Sequence Containers

- <list> είναι διπλά διασυνδεδεμένη λίστα και είναι γρήγορη η εισαγωγή/διαγραφή , αλλά αργή η προσπέλαση
- <deque> είναι ουρά δύο άκρων, δηλαδή εισάγει και διαγράφει στοιχεία από τα δύο άκρα →stack +queue +list



Associative Containers

- Ένα **associative container** είναι **non-sequential** και χρησιμοποιεί ένα κλειδί (*key) για την προσπέλαση των αντικειμένων*. Τα **κλειδιά** είναι **αριθμοί** ή **συμβολοσειρές** χρησιμοποιούνται από τον **container** για να διαχειριστεί τα **αντικείμενα** με μία σειρά , πχ λεξικό.

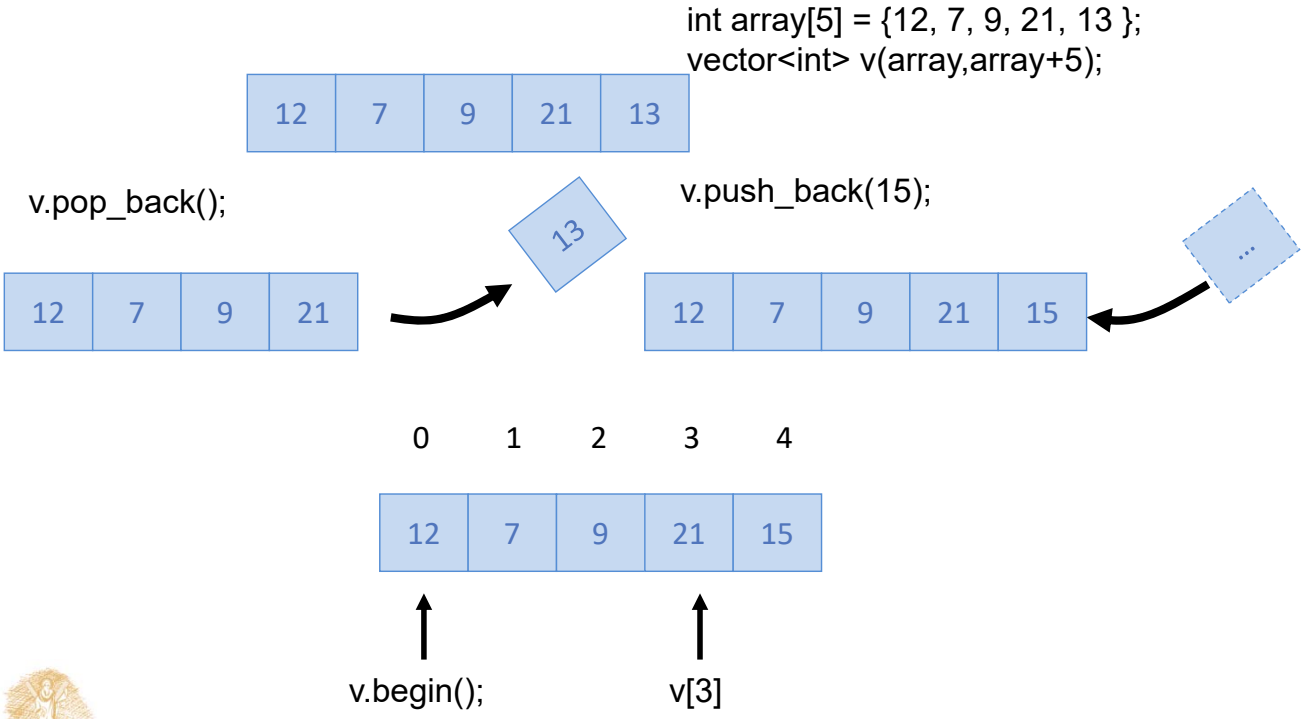
Associative Containers

- Ένα **<set>** αποθηκεύει ένα αριθμό αντικειμένων που περιέχουν κλειδιά. Τα κλειδιά είναι τα χαρακτηριστικά που χρησιμοποιούνται για την διάταξη των στοιχείων.

  Πχ. Ένα set μπορεί να αποθηκεύει αντικείμενα της κλάσης Person που διατάσσονται αλφαβητικά ανάλογα με το όνομα τους

- Ένα **<map>** αποθηκεύει ζευγάρια αντικειμένων : ένα **κλειδί** και μία συσχετιζόμενη **τιμή.** Ένα <map> είναι αντίστοιχο με έναν πίνακα, αλλά αντί για αριθμούς δείκτες μπορούν να χρησιμοποιηθούν οποιοδήποτε τύπου αντικείμενα
- <set> και <map> επιτρέπουν ένα κλειδί για κάθε τιμή, ενώ τα <multiset> και <multimap> επιτρέπουν διπλότυπες εγγραφές (duplicates)

Vector Container



Vector Container

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


vector<int> v(3);

v[0]=23;
v[1]=12;
v[2]=9; // πλήρης


v.push_back(17); // προσθήκη νέου στοιχείου στο τέλος
for (int i=0; i<v.size(); i++)
   cout << v[i] << " "; // random access – στοιχείο i
cout << endl;
```

Vector Container

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

int arr[] = { 12, 3, 17, 8 }; // απλός πίνακας
vector<int> v(arr, arr+4);    // αρχικοποίηση vector από πίνακα

while ( ! v.empty()) {        // Μέχρι να αδειάσει
  cout << v.back() << " ";    // εκτύπωση του τελευταίου στοιχείου
  v.pop_back();               // αφαίρεση του τελευταίου στοιχείου
}
cout << endl;
```

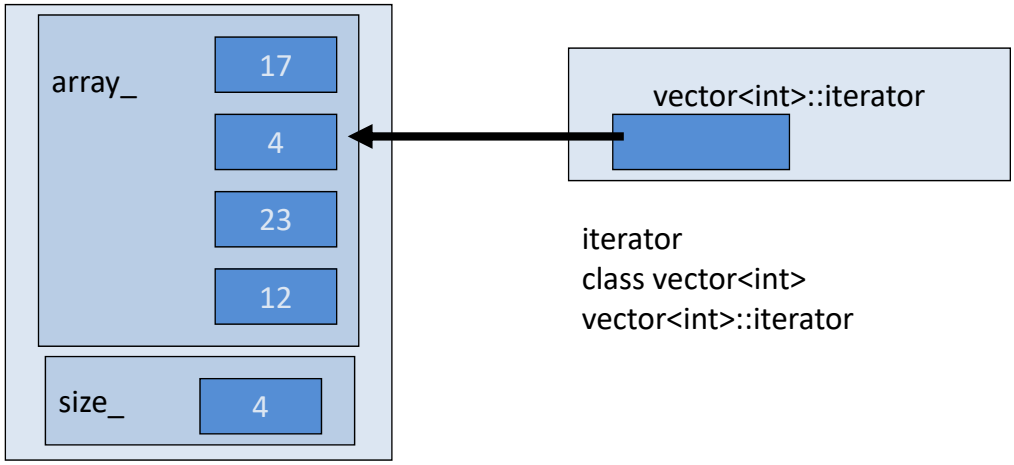
Δημιουργοί Vector

- Ένα vector μπορεί να αρχικοποιηθεί δίνοντας το μέγεθος του και τον τύπο του αντικειμένου (prototype) ή άλλο vector.

```
vector<Date> x(1000); // δημιουργεί vector μεγέθους 1000
                      // απαιτεί ύπαρξη default constructor στην Date

vector<Date> dates(10,Date(17,12,1999));
                      // αρχικοποίηση όλων σε 17.12.1999

vector<Date> y(x);    // αρχικοποίηση του vector y με τον vector x


vector<Date> z(x.begin(), x.end());    // αρχικοποίηση του vector z με
                                       //τον vector x
```

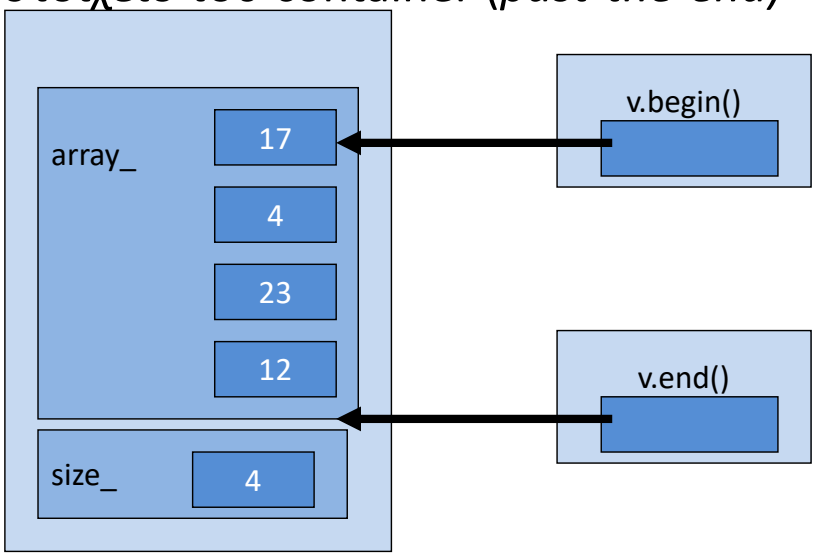
Iterators

- **Iterators** είναι **pointer-like** αντικείμενα που χρησιμοποιούνται για την **προσπέλαση** αντικειμένων σε έναν **container**.
- Χρησιμοποιούνται για να κινούνται σειριακά από αντικείμενο σε αντικείμενο ➔*iterating* διαμέσου ενός container.
- Οι iterators είναι ***member types*** της εκάστοτε κλάσης: typedefs, nested classes…
- Ορίζονται εντός class scope (απαιτείται ::)



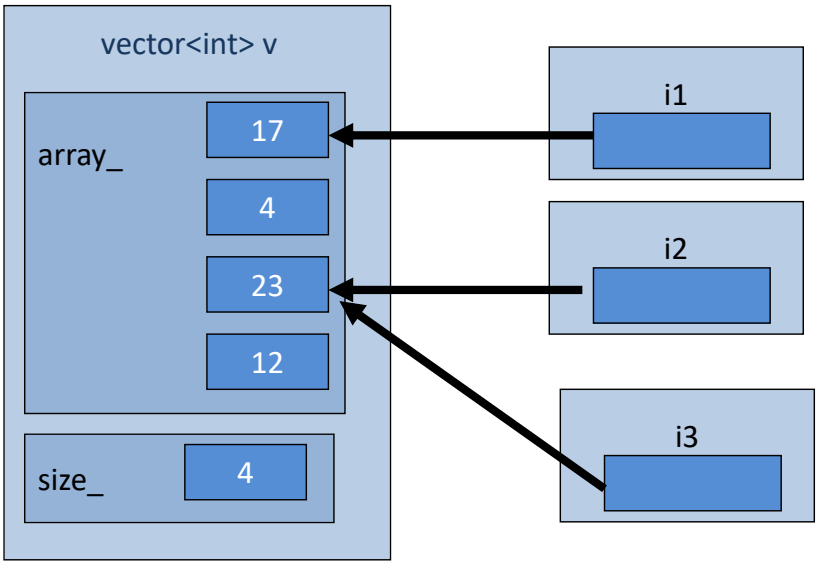
Iterators

- Οι member functions begin() και end() επιστρέφουν έναν iterator στο πρώτο και **META ΤΟ** τελευταίο στοιχείο του container (*past-the-end)*



Iterators

- Μπορούμε να έχουμε πολλαπλούς iterators ταυτόχρονα



Iterators

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

int arr[] = { 12, 3, 17, 8 };
vector<int> v(arr, arr+4);

vector<int>::iterator iter=v.begin(); // δήλωση iterator και ανάθεση ώστε να
                                      //δείχνει στο πρώτο στοιχείο του vector


cout << "first element of v=" << *iter; // Με τον τελεστή * παίρνουμε το
                                        // στοιχείο που δείχνει ο iterator

iter++;                    // ο iterator δείχνει τώρα στο επόμενο στοιχείο

iter=v.end()-1;            // ο iterator δείχνει τώρα στο τελευταίο στοιχείο
```

Iterators

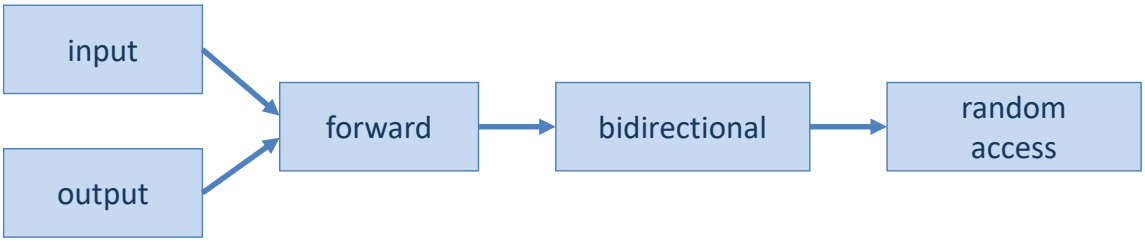
```
int max(vector<int>::iterator start, vector<int>::iterator stop){
  int m=*start;
  while(start != stop) {
    if (*start > m) m=*start;

    ++start;
  }
  return m;
}


cout << "max of v = " << max(v.begin(),v.end());
```

Iterators

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


int arr[] = { 12, 3, 17, 8 };
vector<int> v(arr, arr+4);


for (vector<int>::iterator i=v.begin(); i!=v.end(); i++) {
  cout << *i << " ";
}

cout << endl;
```

Κατηγορίες Iterator

- Δεν είναι δυνατό κάθε iterator να χρησιμοποιηθεί με κάθε container πχ. η κλάση λίστα δεν δίνει random access iterator
- Κάθε αλγόριθμος απαιτεί έναν iterator με κάποιες συγκεκριμένες ιδιότητες πχ. Να χρησιμοποιεί το [] τελεστή για random access
- Οι Iterators χωρίζονται σε 5 κατηγορίες όπου κάθε κατηγορία εμπεριέχει την κατώτερη της πχ. Ένας αλγόριθμος με forward iterator θα δουλεύει και με bidirectional iterator και random access iterator
- Vector και dequeue υποστηρίζουν random access



For_Each() Algorithm

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

void show(int n) {
  cout << n << " ";
}

int arr[] = { 12, 3, 17, 8 };
vector<int> v(arr, arr+4);

for_each (v.begin(), v.end(), show);
   // εφαρμογή της συνάρτησης show
   // σε κάθε στοιχείο του vector v
```

Find() Algorithm

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

int key;
int arr[] = { 12, 3, 17, 8, 34, 56, 9 };
vector<int> v(arr, arr+7);
vector<int>::iterator iter;

cout << "enter value :";
cin >> key;
iter=find(v.begin(),v.end(),key);   // finds integer key in v

if (iter != v.end())
   cout << "Element " << key << " found" << endl;
else
   cout << "Element " << key << " not in vector v" << endl;
```

Find_If() Algorithm

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

Bool mytest(int n) {
  return (n>21) && (n <36);

};


int arr[] = { 12, 3, 17, 8, 34, 56, 9 };
vector<int> v(arr, arr+7);
vector<int>::iterator iter;

iter=find_if(v.begin(),v.end(),mytest);
      // βρίσκει στοιχείο στον v για το οποίον η mytest επιστρέφει true

if (iter != v.end())
   cout << "found " << *iter << endl;
else
   cout << "not found" << endl;
```

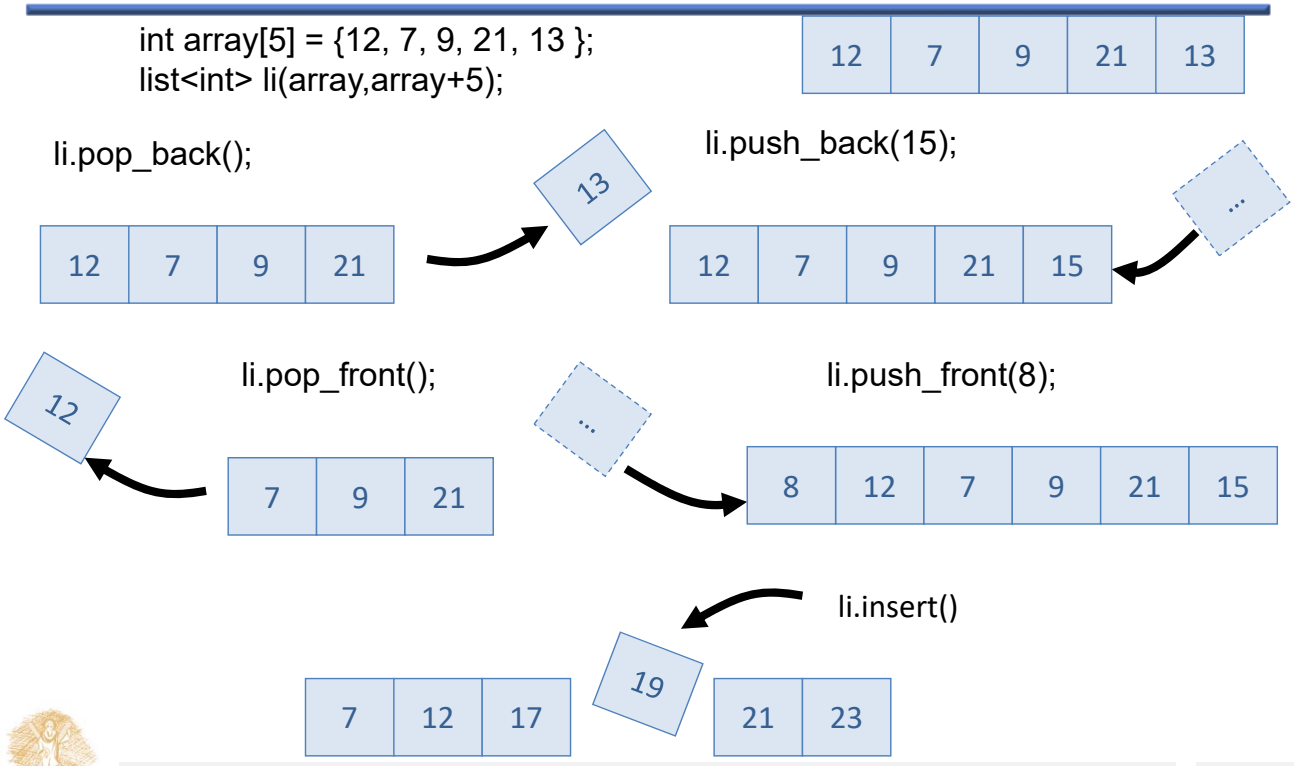
Count_If() Algorithm

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

Bool mytest(int n) {
  return (n>14) && (n <36);
};

int arr[] = { 12, 3, 17, 8, 34, 56, 9 };
 vector<int> v(arr, arr+7);

int n=count_if(v.begin(),v.end(),mytest);
        // μετράει για πόσα στοιχεία του v η mytest επιστρέφει true

cout << "found " << n << " elements" << endl;
```

List Container

- Ένας STL list container είναι μία διπλά διασυνδεδεμένη λίστα: κάθε αντικείμενο δείχνει στον προηγούμενο και στον επόμενο
- Είναι δυνατό να προσθέσουμε/αφαιρέσουμε στοιχεία από τα δύο άκρα της λίστας
- Δεν επιτρέπει random access αλλά μπορούμε να εισάγουμε/διαγράψουμε από την μέση καθώς και να συνενώσουμε και να διατάξουμε

List Container



Insert Iterators

- Η χρήση του copy algorithm διαγράφει τα υπάρχοντα στοιχεία

```
#include <list>
int arr1[]= { 1, 3, 5, 7, 9 };
int arr2[]= { 2, 4, 6, 8, 10 };
list<int> l1(arr1, arr1+5);
list<int> l2(arr2, arr2+5);
copy(l1.begin(), l1.end(), l2.begin());
// αντιγραφή των στοιχείων του l1 στο l2 (τα στοιχεία του l2 διαγράφονται)
// l2 = { 1, 3, 5, 7, 9 }
```

Insert Iterators

- Οι iterators διαφοροποιούν τη συμπεριφορά του copy algorithm
  - back_inserter : εισάγει νέο στοιχείο στο τέλος
  - front_inserter : εισάγει νέο στοιχείο στην αρχή
  - inserter : εισάγει νέο στοιχείο σε θέση

```
#include <list>
int arr1[]= { 1, 3, 5, 7, 9 };
int arr2[]= { 2, 4, 6, 8, 10 };
list<int> l1(arr1, arr1+5);
list<int> l2(arr2, arr2+5);
copy(l1.begin(), l1.end(), back_inserter(l2));   // χρήση back_inserter
      // adds contents of l1 to the end of l2 = { 2, 4, 6, 8, 10, 1, 3, 5, 7, 9 }
copy(l1.begin(), l1.end(), front_inserter(l2));  // use front_inserter
      // adds contents of l1 to the front of l2 = { 1, 3, 5, 7, 9, 2, 4, 6, 8, 10 }
copy(l1.begin(), l1.end(), inserter(l2,l2.begin()));
      // adds contents of l1 at the "old" beginning of l2 = { 1, 3, 5, 7, 9, 2, 4, 6, 8, 10 }
```

Sort & Merge

- Sort και merge

```
#include <list>

int arr1[]= { 6, 4, 9, 1, 7 };
int arr2[]= { 4, 2, 1, 3, 8 };
list<int> l1(arr1, arr1+5);
list<int> l2(arr2, arr2+5);
l1.sort();    // l1 = {1, 4, 6, 7, 9}
l2.sort();    // l2= {1, 2, 3, 4, 8 }
l1.merge(l2); // merges l2 into l1
       // l1 = { 1, 1, 2, 3, 4, 4, 6, 7, 8, 9}, l2= {}
```

Functions Objects

- sort, merge, accumulate μπορούν να πάρουν σαν ορίσματα ένα function object
- Το function object είναι ένα αντικείμενο μιας template class που έχει κάνει overload τον τελεστή κλήσης συνάρτησης **operator()**
- Είναι δυνατό να χρησιμοποιηθούν user-written functions στη θέση των pre-defined function objects

```
#include <list> #include <functional>
int arr1[]= { 6, 4, 9, 1, 7 };
list<int> l1(arr1, arr1+5);

l1.sort(greater<int>());    // χρήση function object greater<int>
        // για ταξινόμηση αντίστροφης σειράς l1 = { 9, 7, 6, 4, 1 }
```

Function Objects

- Ο algorithm accumulate συναθροίζει δεδομένα πάνω από τα στοιχεία πχ sum of elements

```
#include <list>
#include <functional>
#include <numeric>

int arr1[]= { 6, 4, 9, 1, 7 };
list<int> l1(arr1, arr1+5); // αρχικοποίηση l1 απο arr1

int sum = accumulate(l1.begin(), l1.end(),0, plus<int>());
int sum = accumulate(l1.begin(), l1.end(),0); // ισοδύναμο
int fac = accumulate(l1.begin(), l1.end(),0, times<int>());
```

User Defined Function Objects

```
class squared_sum{
   public:
      int operator()(int n1, int n2) {
         return n1+n2*n2;
      }
};

int sq = accumulate(l1.begin(), l1.end() , 0, squared_sum() );
                    // computes the sum of squares
```

User Defined Function Objects

```
template <class T>
class squared_sum {
  public:
    T operator()(T n1, T n2) {
      return n1+n2*n2;
    }
};

vector<complex> vc;
complex sum_vc;
vc.push_back(complex(2,3));
vc.push_back(complex(1,5));
vc.push_back(complex(-2,4));
sum_vc = accumulate(vc.begin(), vc.end(),complex(0,0),squared_sum<complex>() );
                // computes the sum of squares of a vector of complex numbers
```

Associative Containers

- Σε ένα associative container τα αντικείμενα δεν σχηματίζουν ακολουθία αλλά δενδρικές δομές ή hash table.
- Τα προτερήματά τους είναι η ταχύτητα αναζήτησης (δυαδική κλπ)
- Η αναζήτηση γίνεται με βάση το κλειδί που είναι είτε αριθμός είτε συμβολοσειρά
- Η *value* είναι ένα χαρακτηριστικό των objects στο container
- Η STL έχει δύο βασικά associative containers
  - sets and multisets
  - maps and multimaps

Sets and Multisets

```
#include <set>
string names[] = {"Yiannis", "Takis", "Petros", "Christos", "Anestis"};

set<string, less<string> > nameSet(names,names+5);
                // δημιουργία σετ με ονόματα αλφαβητικά ταξινομημένα

nameSet.insert("Eleni"); // εισαγωγή νέων στοιχείων
nameSet.insert("Maria");
nameSet.erase("Petros"); // αφαίρεση στοιχείου
set<string, less<string> >::iterator iter; // set iterator
string searchname;
cin >> searchname;
iter=nameSet.find(searchname); // εύρεση του ονόματος στο σετ

if (iter == nameSet.end()) // έλεγχος αν ο iterator δείχνει στο τέλος
  cout << searchname << " not in set!" <<endl;
else
  cout << searchname << " is in set!" <<endl;
```

Set and Multisets

```
string names[]={"Yiannis","Takis","Petros","Christos","Anestis"};

set<string, less<string> > nameSet(names,names+5);
set<string, less<string> >::iterator iter; // set iterator

iter=nameSet.lower_bound("K");
      // set iterator to lower start value "K"

while (iter != nameSet.upper_bound("Z"))
   cout << *iter++ << endl;
// displays Petros, Takis, Yiannis
```

Maps and Multimaps

- Ένα map αποθηκεύει pairs <key, value> ενός key object και ενός associated value object.
- Το key object περιέχει ένα key με το οποίο αναζητούμε και το value object περιέχει τις πληροφορίες
- Το key μπορεί να είναι string, πχ όνομα ή αριθμός

Maps and Multimaps

```
#include <map>
string names[]= {"Yiannis", "Takis", "Petros", "Christos", "Anestis"};
int numbers[]= {62622, 56879, 12587, 387546, 98794};

map<string, int, less<string> > phonebook;

map<string, int, less<string> >::iterator iter;

for (int j=0; j<5; j++)
  phonebook[names[j]]=numbers[j]; // αρχικοποίηση

for (iter = phonebook.begin(); iter !=phonebook.end(); iter++)
  cout << (*iter).first << " : " << (*iter).second << endl;
  cout << "Phone of Yiannis is: " << phonebook["Yiannis"] << endl;
```

Person Class

```
Class Person {
  private: string lastName;
    string firstName;
    long phoneNumber;

  public:
    Person(string lana, string fina, long pho) : lastName(lana),
                              firstName(fina), phonenumber(pho) {}

    bool operator<(const Person& p);
    bool operator==(const Person& p);
}
```

multiset

```
Person p1("John", "Doe", 284545632);
Person p2("Mark", "Smith",226874521);
Person p3("Peter", "Fontain", 284586541);

multiset< Person, less< Person >> personSet;
multiset< Person, less< Person >>::iterator iter;

personSet.insert(p1);
personSet.insert(p2);
personSet.insert(p3);
```