STL2 - plan

- Algoritmi în STL
  - Operații ce nu modifică secvența
  - Operații ce modifică secvența
  - Sortare
  - Căutare binară
  - Merge
  - Heap
  - Min/max
- Exemple

http://www.cplusplus.com/reference/algorithm/

Standard Template Library: Algorithms

- Header-ul <algorithm> definește o colecție de funcții proiectate pentru a fi aplicate unui domeniu de elemente ale unui container

- Domeniul (range) este orice secvență de obiecte, dintr-un tablou sau container, ce pot fi accesate via pointer sau iterator

- Algoritmii implementați de aceste funcții operează direct pe valorile pointate, nu sunt afectate în nici un fel structura containerului (dimensiune, memoria alocată) .

Operații ce nu modifică secvența

• for_each	Apply function to range
• find	Find value in range
• find_if	Find element in range
• find_end	Find last subsequence in range
• find_first_of	Find element from set in range
• adjacent_find	Find equal adjacent elements in range
• count	Count appearances of value in range
• count_if	Return number of elements in range satisfying condition
• mismatch	Return first position where two ranges differ
• equal	Test whether the elements in two ranges are equal
• search	Find subsequence in range
• search_n	Find succession of equal values in range

Exemple

```
void myfunction (int i) {
  cout << " " << i;
}
struct myclass {
  void operator() (int i) {cout << " " << i;}
} myobject;
int main () {
  vector<int> myvector;
  myvector.push_back(10);
  myvector.push_back(20);
  myvector.push_back(30);
  cout << "myvector contains:";
  for_each (myvector.begin(), myvector.end(), myfunction);

  // or:
  cout << "\nmyvector contains:";
  for_each (myvector.begin(), myvector.end(), myobject);

  cout << endl;

  return 0;
}
```

Exemple

```
vector<int> myvector;
for (int i=1; i<10; i++) myvector.push_back(i); // 1 2 3 4 5 6 7 8 9

mycount = (int) count_if (myvector.begin(), myvector.end(), IsOdd);
cout << "myvector contains " << mycount  << " odd values.\n";
```

```
    int myints[]={10,20,30,30,20,10,10,20};
    vector<int> myvector (myints,myints+8);

    vector<int>::iterator it;
    it = search_n (myvector.begin(), myvector.end(), 2, 30); // 2 (== 30)
    it = search_n (myvector.begin(), myvector.end(), 2, 10, mypredicate);
```

Operații ce modifică secvența(1)

copy	Copy range of elements
copy_backward	Copy range of elements backwards
swap	Exchange values of two objects
swap_ranges	Exchange values of two ranges
iter_swap	Exchange values of objects pointed by two iterators
transform	Apply function to range
replace	Replace value in range
replace_if	Replace values in range
replace_copy	Copy range replacing value
replace_copy_if	Copy range replacing value
fill	Fill range with value
fill_n	Fill sequence with value
generate	Generate values for range with function
generate_n	Generate values for sequence with function

Exemple

```
int myints[]={10,20,30,40,50,60,70};
vector<int> myvector;
vector<int>::iterator it;
myvector.resize(7);
copy ( myints, myints+7, myvector.begin() );

int RandomNumber () { return (rand()%100); }
struct c_unique {
   int current; c_unique() {current=0;}
   int operator()() {return ++current;}
} UniqueNumber

vector<int> myvector (8);
  vector<int>::iterator it;
  generate (myvector.begin(), myvector.end(), RandomNumber);
                                              // functie generator
  cout << "myvector contains:";
  for (it=myvector.begin(); it!=myvector.end(); ++it)
    cout << " " << *it;
  generate (myvector.begin(), myvector.end(), UniqueNumber);
                   // obiect dintr-o clasa cu operator()
```

Exemple

```
int myints[]={10,20,30,40,50 };            //   myints:  10  20  30  40  50
vector<int> myvector (4,99);               // myvector:  99  99  99  99

iter_swap(myints,myvector.begin());        //   myints: [99] 20  30  40  50
                                           // myvector: [10] 99  99  99

iter_swap(myints+3,myvector.begin()+2);    //   myints:  99  20  30 [99]
                                           // myvector:  10  99 [40] 99



vector<int> myvector (8);                      // myvector: 0 0 0 0 0 0 0 0

fill (myvector.begin(),myvector.begin()+4,5);// myvector: 5 5 5 5 0 0 0 0
fill (myvector.begin()+3,myvector.end()-2,8);// myvector: 5 5 5 8 8 8 0 0
```

Operații ce modifică secvența(2)

• remove	Remove value from range
• remove_if	Remove elements from range
• remove_copy	Copy range removing value
• remove_copy_if	Copy range removing values
• unique	Remove consecutive duplicates in range
• unique_copy	Copy range removing duplicates
• reverse	Reverse range
• reverse_copy	Copy range reversed
• rotate	Rotate elements in range
• rotate_copy	Copy rotated range
• random_shuffle	Rearrange elements in range randomly
• partition	Partition range in two
• stable_partition	Partition range in two - stable ordering

Exemple

```
template <class ForwardIterator> void rotate ( ForwardIterator first,
   ForwardIterator middle, ForwardIterator last );

   for (int i=1; i<10; ++i) myvector.push_back(i); // 1 2 3 4 5 6 7 8 9

   rotate(myvector.begin(),myvector.begin()+3,myvector.end());
                                                   // 4 5 6 7 8 9 1 2 3


  vector<int> myvector;
  for (int i=1; i<10; ++i) myvector.push_back(i); // 1 2 3 4 5 6 7 8 9

  reverse(myvector.begin(),myvector.end());       // 9 8 7 6 5 4 3 2 1


  for (int i=1; i<10; ++i) myvector.push_back(i); // 1 2 3 4 5 6 7 8 9
  bound = partition (myvector.begin(), myvector.end(), IsOdd);

// se rearanjeaza elementele vectorului
// bound = adresa de inceput a partii a doua
// 1 9 3 7 5 6 4 8 2
```

Sortare

• sort	Sort elements in range
• stable_sort	Sort elements preserving order of equivalents
• partial_sort	Partially Sort elements in range
• partial_sort_copy	Copy and partially sort range
• nth_element	Sort element in range

Exemple

```
template <class RandomAccessIterator>
  void sort ( RandomAccessIterator first, RandomAccessIterator last );

template <class RandomAccessIterator, class Compare>
  void sort ( RandomAccessIterator first, RandomAccessIterator last,
   Compare comp );

vector<int> myvector (myints, myints+8);        // 32 71 12 45 26 80 53 33
  vector<int>::iterator it;

  // using default comparison (operator <):
  sort (myvector.begin(), myvector.begin()+4);  //(12 32 45 71)26 80 53 33

  // using function as comp
  sort (myvector.begin()+4, myvector.end(), myfunction);
                         // 12 32 45 71(26 33 53 80)

  // using object as comp
  sort (myvector.begin(), myvector.end(), myobject);
                         //(12 26 32 33 45 53 71 80)
```

Exemple

```
  double mydoubles[] = {3.14, 1.41, 2.72, 4.67, 1.73, 1.32, 1.62, 2.58};
  vector<double> myvector;
  vector<double>::iterator it;


bool compare_as_ints (double i,double j)
{
  return (int(i)<int(j));
}

  myvector.assign(mydoubles,mydoubles+8);
  stable_sort (myvector.begin(), myvector.end());
  // 1.32 1.41 1.62 1.73 2.58 2.72 3.14 4.67

  myvector.assign(mydoubles,mydoubles+8);
  stable_sort (myvector.begin(), myvector.end(), compare_as_ints);
  // elementele cu aceeasi valoare raman in aceeasi ordine
  // 1.41 1.73 1.32 1.62 2.72 2.58 3.14 4.67
```

Exemple

```
  int myints[] = {9,8,7,6,5,4,3,2,1};
  vector<int> myvector (myints, myints+9);
  vector<int>::iterator it;

  // using default comparison (operator <):
  partial_sort (myvector.begin(), myvector.begin()+5, myvector.end());

  // using function as comp
  partial_sort (myvector.begin(), myvector.begin()+5,
   myvector.end(),myfunction);

//1 2 3 4 5 9 8 7 6

  vector<int> myvector(15);
  generate (myvector.begin(), myvector.end(), RandomNumber);
//41 67 34 0 69 24 78 58 62 64 5 45 81 27 61

  partial_sort (myvector.begin(), myvector.begin()+5, myvector.end());
//0 5 24 27 34 69 78 67 62 64 58 45 81 41 61
```

Căutare binară (pe secvențe sortate):

• **lower_bound**	Return iterator to lower bound
• **upper_bound**	Return iterator to upper bound
• **equal_range**	Get subrange of equal elements
• **binary_search**	Test if value exists in sorted array

Exemple

```
  int myints[] = {10,20,30,30,20,10,10,20};
  vector<int> v(myints,myints+8);            // 10 20 30 30 20 10 10 20
  vector<int>::iterator low,up;

  sort (v.begin(), v.end());                 // 10 10 10 20 20 20 30 30

  low=lower_bound (v.begin(), v.end(), 20); //  3
  up= upper_bound (v.begin(), v.end(), 20); //  6

  bounds=equal_range (v.begin(), v.end(), 20); // 3 6


template <class ForwardIterator, class T>
  bool binary_search ( ForwardIterator first, ForwardIterator last,
                       const T& value );

template <class ForwardIterator, class T, class Compare>
  bool binary_search ( ForwardIterator first, ForwardIterator last,
                       const T& value, Compare comp );

   if (binary_search (v.begin(), v.end(), 3)) //…
   if (binary_search (v.begin(), v.end(), 6, myfunction)) // …
```

Merge (pe secvențe sortate)

• **merge**	Merge sorted ranges
• **inplace_merge**	Merge consecutive sorted ranges
• **includes**	Test whether sorted range includes another sorted range
• **set_union**	Union of two sorted ranges
• **set_intersection**	Intersection of two sorted ranges
• **set_difference**	Difference of two sorted ranges
• **set_symmetric_difference**	Symmetric difference of two sorted ranges

Exemple

```
template <class InputIterator1, class InputIterator2, class OutputIterator>
OutputIterator merge ( InputIterator1 first1, InputIterator1 last1,
                       InputIterator2 first2, InputIterator2 last2,
                       OutputIterator result );

template <class InputIterator1, class InputIterator2,
        class OutputIterator, class Compare>
OutputIterator merge ( InputIterator1 first1, InputIterator1 last1,
                       InputIterator2 first2, InputIterator2 last2,
                       OutputIterator result, Compare comp );


int first[] = {5,10,15,20,25};
int second[] = {50,40,30,20,10};
vector<int> v(10);

sort (first,first+5);   //  5,10,15,20,25
sort (second,second+5); // 10,20,30,40,50
merge (first,first+5,second,second+5,v.begin());
//5 10 10 15 20 20 25 30 40 50
```

Heap

• **push_heap**	Push element into heap range
• **pop_heap**	Pop element from heap range
• **make_heap**	Make heap from range
• **sort_heap**	Sort elements of heap

Exemple

```
template <class RandomAccessIterator>
void make_heap ( RandomAccessIterator first, RandomAccessIterator last );

template <class RandomAccessIterator, class Compare>
void make_heap ( RandomAccessIterator first, RandomAccessIterator last,
                 Compare comp );

int myints[] = {10,20,30,5,15};
vector<int> v(myints,myints+5);

make_heap (v.begin(),v.end());
cout << "initial max heap   : " << v.front() << endl; // 30

pop_heap (v.begin(),v.end()); v.pop_back();
cout << "max heap after pop : " << v.front() << endl; // 20

v.push_back(99); push_heap (v.begin(),v.end());
cout << "max heap after push: " << v.front() << endl; // 99

sort_heap (v.begin(),v.end());
cout << "final sorted range :";
for (unsigned i=0; i<v.size(); i++) cout << " " << v[i]; // 5 10 15 20 99
```

Min/max

• **min**	Return the lesser of two arguments
• **max**	Return the greater of two arguments
• **min_element**	Return smallest element in range
• **max_element**	Return largest element in range
• **lexicographical_compare**	Lexicographical less-than comparison
• **next_permutation**	Transform range to next permutation
• **prev_permutation**	Transform range to previous permutation

Exemple

```
template <class ForwardIterator>
ForwardIterator min_element ( ForwardIterator first, ForwardIterator last );

template <class ForwardIterator, class Compare>
ForwardIterator min_element ( ForwardIterator first, ForwardIterator last,
                              Compare comp );

int myints[] = {3,7,2,5,6,4,9};
// using default comparison:
cout << *min_element(myints,myints+7) << endl; // 2
cout << *max_element(myints,myints+7) << endl; // 9

char first[]="Apple";         // 5 letters
char second[]="apartment";    // 9 letters
cout << "Using default comparison (operator<): ";
if (lexicographical_compare(first,first+5,second,second+9))
  cout << first << " is less than " << second << endl;
else
  if (lexicographical_compare(second,second+9,first,first+5))
    cout << first << " is greater than " << second << endl;
else
  cout << first << " and " << second << " are equivalent\n";
```

Exemple

```
template <class BidirectionalIterator>
bool next_permutation (BidirectionalIterator first,
                       BidirectionalIterator last );

template <class BidirectionalIterator, class Compare>
bool next_permutation (BidirectionalIterator first,
                       BidirectionalIterator last, Compare comp);


int myints[] = {1,2,3};
cout << "The 3! possible permutations with 3 elements:\n“;
sort (myints,myints+3);
do {
  cout << myints[0] << " " << myints[1] << " " << myints[2] << endl;
} while ( next_permutation (myints,myints+3) );
```