

11

Generic Programming (*with STL in C++*)

Standard Template Library

Nesneye dayalı programlamada, verinin birincil öneme sahip programlama birimi olduğunu belirtmiştik. Veri, fiziksel yada soyut bir çok büyüklüğü modelleyebilir. Bu model oldukça basit yada karmaşık olabilir. Her nasıl olursa olsun, veri mutlaka bellekte saklanmaktadır ve veriye benzer biçimlerde erişilmektedir. C++, oldukça karmaşık veri tiplerini ve yapılarını oluşturmamıza olanak sağlayan mekanizmalara sahiptir. Genel olarak, programların, bu veri yapılarına belirli bazı biçimlerde eriştiğini biliyoruz:

`array`, `list`, `stack`, `queue`, `vector`, `map`, ...

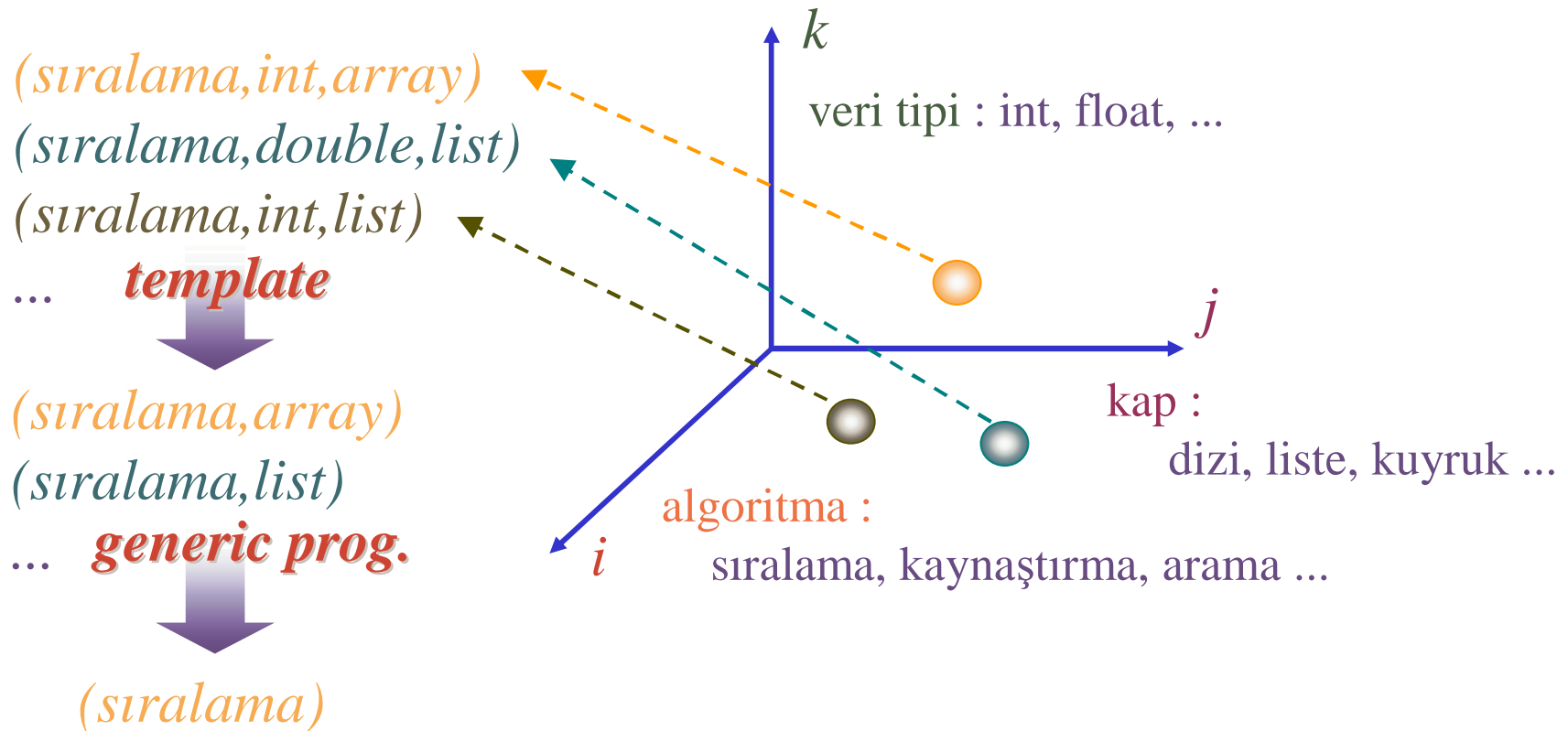
STL kütüphanesi verinin bellekteki organizasyonuna, erişimine ve işlenmesine yönelik çeşitli yöntemler sunmaktadır. Bu bölümde bu yöntemleri inceleyeceğiz.

Standard Template Library (STL) Hewlett Packard'ın Palo Alto (California)'daki laboratuvarlarında Alexander Stepanov ve Meng Lee tarafından geliştirilmiştir.

1970'lerin sonlarında Alexander Stepanov bir kısım algoritmaların veri yapısının nasıl depolandıklarından bağımsız olduklarını gözlemledi. Örneğin, sıralama algoritmalarında sıralanacak sayıların bir dizide mi? yoksa bir listede mi? bulunduğu bir önemi yoktur. Değişen sadece bir sonraki elemana nasıl erişildiği ile ilgilidir. Stepanov bu ve benzeri algoritmaları inceleyerek, algoritmaları veri yapısından bağımsız olarak performanstan ödün vermeksizin soyutlamayı başarmıştır. Bu fikrini 1985'de Generic ADA dilinde gerçekleştirmiştir. Ancak o dönemde henüz C++'da bir önceki bölümde incelediğimiz Template yapısı bulunmadığı için bu fikrini C++'da ancak 1992 yılında gerçekleştirebilmiştir.

Generic Programming

Bir yazılım ürününün bileşenlerini, üç boyutlu uzayda bir nokta olarak düşünebiliriz :



STL Bileşenleri

STL üç temel bileşenden oluşmaktadır:

- **Algoritma,**
- **Kap (Container):** nesnelere depolamak ve yönetmekten sorumlu nesne,
 - Lineer Kaplar : Vector, Deque, List
 - Asosyatif Kaplar : Set, Map, Multi-set, Multi-map
- **Yineleyici (Iterator):** algoritmanın farklı tipte kaplarla çalışmasını sağlayacak şekilde erişimin soyutları.

Kaplar

| Container | Description | Required Header |
|----------------|---|-----------------|
| bitset | A set of bits | <bitset> |
| deque | A double-ended queue | <deque> |
| list | A linear list | <list> |
| map | Stores key/value pairs in which each key is associated with only one value | <map> |
| multimap | Stores key/value pairs in which one key may be associated with two or more values | <map> |
| multiset | A set in which each element is not necessarily unique | <set> |
| priority_queue | A priority queue | <queue> |
| queue | A queue | <queue> |
| set | A set in which each element is unique | <set> |
| stack | A stack | <stack> |
| vector | A dynamic array | <vector> |

C++'da sabit boyutlu dizi tanımlamak yürütme zamanında belleğin ya kötü kullanılmasına yada dizi boyunun yetersiz kalmasına neden olmaktadır.

STL kütüphanesindeki **vector** kabı bu sorunları gidermektedir.

STL kütüphanesindeki **list** kabı, bağlantılı liste yapısıdır.

deque (*Double-Ended QUEUE*) kabı, yığın ve kuyruk yapılarının birleşimi olarak düşünülebilir. deque kabı her iki uçtan veri eklemeye ve silmeye olanak sağlamaktadır.

| | | |
|---------------|--|---|
| Vector | Relocating, expandable array Slow to insert or erase in the middle. | Quick random access (by index number). Quick to insert or erase at end. |
| List | Doubly linked list Quick access to both ends. | Quick to insert or delete at any location. Slow random access. |
| Deque | Like vector, but can be accessed at either end Slow to insert or erase in the middle. | Quick random access (using index number). Quick to insert or erase (push and pop) at either the beginning or the end. |

Vector

▶ `#include <vector>`

▶ Kurucular

- Boş: `vector<string> object;`
- Belirli sayıda eleman:
 - `vector<string> object(5,string(“hello”)) ;`
 - `vector<string> container(10)`
 - `vector<string> object(&container[5], &container[9]);`
 - `vector<string> object(container) ;`

Vector Member Functions

- ▶ `Type &vector::back()`: returns the reference to the last element
- ▶ `Type &vector::front()`: returns the reference to the first element
- ▶ `vector::iterator vector::begin()`
- ▶ `vector::iterator vector::end()`
- ▶ `vector::clear()`
- ▶ `bool vector::empty()`
- ▶ `vector::iterator vector::erase()`
 - `erase(pos)`
 - `erase(first,beyond)`

Vector Member Functions

- ▶ `vector::insert`
 - `vector::iterator insert(pos)`
 - `vector::iterator insert(pos,value)`
 - `vector::iterator insert(pos,first,beyond)`
 - `vector::iterator insert(pos,n,value)`
- ▶ `void vector::pop_back()`
- ▶ `void vector::push_back(value)`
- ▶ `vector::resize()`
 - `resize(n,value)`
- ▶ `vector::swap()`
 - `vector<int> v1(7),v2(10) ;`
 - `v1.swap(v2);`
- ▶ `unsigned vector::size()`

Örnek

```
vector<int> v;  
cout << v.capacity() << v.size() ;  
v.insert(v.end(),3) ;  
cout << v.capacity() << v.size() ;  
v.insert (v.begin(), 2, 5);
```

v = (3)

v = (5,5,3)

```
vector<int> w (4,9);  
w.insert(w.end(), v.begin(), v.end() );  
w.swap(v) ;
```

w = (9,9,9,9)

w = (9,9,9,9,5,5,3)

v = (9,9,9,9,5,5,3)

w=(5,5,3)

```
w.erase(w.begin());  
w.erase(w.begin(),w.end() );  
cout << w.empty() ? "Empty" : "not Empty"
```

w = (5,3)

Empty

```
#define __USE_STL
```

```
// STL include files
```

```
#include <vector>
```

```
#include <list>
```

```
vector<int> v;
```

```
v.insert(v.end(),3);
```

v = (3)

```
v.insert(v.begin(),5);
```

v = (5,3)

```
cout << v.front() << endl;
```

5

```
cout << v.back();
```

3

```
v.pop_back();
```

```
cout << v.back();
```

5

List

- ▶ `#include <list>`
- ▶ Eklenen eleman sayısı belirli olmadığı durumlarda uygundur
- ▶ Kurucular
 - Boş: `list<string> object;`
 - Belirli sayıda eleman:
 - `list<string> object(5,string("hello")) ;`
 - `list<string> container(10)`
 - `list<string> object(&container[5], &container[9]);`
 - `list<string> object(container) ;`

List Member Functions

- ▶ Type `&list::back()`: returns the reference to the last element
- ▶ Type `&list::front()`: returns the reference to the first element
- ▶ `list::iterator list::begin()`
- ▶ `list::iterator list::end()`
- ▶ `list::clear()`
- ▶ `bool list::empty()`
- ▶ `list::iterator list::erase()`
 - `erase(pos)`
 - `erase(first,beyond)`

List Member Functions

- ▶ `list::insert`
 - `list::iterator insert(pos)`
 - `list::iterator insert(pos,value)`
 - `list::iterator insert(pos,first,beyond)`
 - `list::iterator insert(pos,n,value)`
- ▶ `void list::pop_back()`
- ▶ `void list::push_back(value)`
- ▶ `list::resize()`
 - `resize(n,value)`
- ▶ `void list<type>::merge(list<type> other)`
- ▶ `void list<type>::remove(value)`
- ▶ `unsigned list::size()`



list2.cpp



list1.cpp

List Member Functions

- ▶ `list::sort()`
- ▶ `void list::splice(pos,object)`
- ▶ `void list::unique()`: operates on sorted list, removes consecutive identical elements



`list3.cpp`



`list4.cpp`

Queue

- ▶ `#include <queue>`
- ▶ FIFO (=First In First Out)
- ▶ Kurucular
 - Boş: `queue<string> object;`
 - Kopya Kurucu: `queue<string> object(container) ;`

Queue Member Functions

- ▶ `Type &queue::back()`: returns the reference to the last element
- ▶ `Type &queue::front()`: returns the reference to the first element
- ▶ `bool queue::empty()`
- ▶ `void queue::push(value)`
- ▶ `void queue::pop()`

Priority_Queue

- ▶ `#include <queue>`
- ▶ Temel olarak `queue` ile aynı
- ▶ Kuyruğa ekleme belirli bir önceliğe göre yürütülür
- ▶ Öncelik: `operator<()`



`priqueue1.cpp`



`priqueue2.cpp`

Priority_Queue Member Functions

- ▶ `Type &queue::back()`: returns the reference to the last element
- ▶ `Type &queue::front()`: returns the reference to the first element
- ▶ `bool queue::empty()`
- ▶ `void queue::push(value)`
- ▶ `void queue::pop()`

Deque

- ▶ `#include <deque>`
- ▶ Head & Tail, Doubly Linked
- ▶ `deque<string> object`
 - `deque<string> object(5, string("hello")) ;`
 - `deque<string> container(10)`
 - `deque<string> object(&container[5], &container[9]);`
 - `deque<string> object(container) ;`

Deque Member Functions

- ▶ Type `&deque::back()`: returns the reference to the last element
- ▶ Type `&deque::front()`: returns the reference to the first element
- ▶ `deque::iterator deque::begin()`
- ▶ `deque::iterator deque::end()`
- ▶ `deque::clear()`
- ▶ `bool deque::empty()`
- ▶ `deque::iterator deque::erase()`
 - `erase(pos)`
 - `erase(first,beyond)`

Deque Member Functions

- ▶ `vector::insert`
 - `deque::iterator insert(pos)`
 - `deque::iterator insert(pos,value)`
 - `deque::iterator insert(pos,first,beyond)`
 - `deque::iterator insert(pos,n,value)`
- ▶ `void deque::pop_back()`
- ▶ `void deque::push_back(value)`
- ▶ `deque::resize()`
 - `resize(n,value)`
- ▶ `deque::swap()`
- ▶ `unsigned deque::size()`

Asosyatif Kaplar: Set, Multiset, Map, Multimap

- Set sıralı küme oluşturmak için kullanılır.

```
#include <set>
```

```
using namespace std;
```

```
int main(){
```

```
    string names[] = {"Katie", "Robert", "Mary", "Amanda", "Marie"};
```

```
    set<string> nameSet(names, names+5); // initialize set to array
```

```
    set<string>::const_iterator iter; // iterator to set
```

```
    nameSet.insert("Jack");           // insert some more names
```

```
    nameSet.insert("Larry");
```

```
    nameSet.insert("Robert");        // no effect; already in set
```

```
    nameSet.insert("Barry");
```

```
    nameSet.erase("Mary");           // erase a name
```



```
cout << "\nSize=" << nameSet.size() << endl;
iter = nameSet.begin();           // display members of set
while( iter != nameSet.end() )
    cout << *iter++ << '\n';
string searchName;                // get name from user
cout << "\nEnter name to search for: ";
cin >> searchName;               // find matching name in set
iter = nameSet.find(searchName);
if( iter == nameSet.end() )
    cout << "The name" << searchName << " is NOT in the set.";
else
    cout << "The name " << *iter << " IS in the set.";
}
```

```
// set2.cpp set
int main() {
    set<string> city;
    set<string>::iterator iter;
    city.insert("Trabzon");    // insert city names
    city.insert("Adana");
    city.insert("Edirne");
    city.insert("Bursa");
    city.insert("Istanbul");
    city.insert("Rize");
    city.insert("Antalya");
    city.insert("Izmir");
    city.insert("Hatay");
    city.insert("Ankara");
    city.insert("Zonguldak");
```

```
iter = city.begin();          // display set
while( iter != city.end() )
    cout << *iter++ << endl;
```

```
string lower, upper;        // display entries in range
cout << "\nEnter range (example A Azz): ";
cin >> lower >> upper;
iter = city.lower_bound(lower);
while( iter != city.upper_bound(upper) )
    cout << *iter++ << endl;
}
```

Map

- ▶ #include <map>
- ▶ Key/Value pairs
- ▶ map<string,int> object
 - pair<string,int>

```
pa[]= {  
    pair<string,int>("one",1),  
    pair<string,int>("two",2),  
    pair<string,int>("three",3),  
    pair<string,int>("four",4)  
};
```
 - map<string,int> object(&pa[0],&pa[3]);
- ▶ object["two"]

Map Member Functions

- ▶ `map::insert`
 - `pair<map::iterator,bool> insert(keyvalue)`
 - `pair<map::iterator,bool> insert(pos,keyvalue)`
 - `void insert(first,beyond)`
- ▶ `map::iterator map::lower_bound(key)`
- ▶ `map::iterator map::upper_bound(key)`
- ▶ `pair<map::iterator,map::iterator> map::equal_range(key)`
- ▶ `map::iterator map::find(key)`
 - returns `map::end()` if not found
- ▶ `unsigned deque::size()`

Map Member Functions

- ▶ `map::iterator map::begin()`
- ▶ `map::iterator map::end()`
- ▶ `map::clear()`
- ▶ `bool map::empty()`
- ▶ `map::iterator map::erase()`
 - `erase(keyvalue)`
 - `erase(pos)`
 - `erase(first,beyond)`

Örnek

```
int main(){
    map<string,int> city_num;
    city_num["Trabzon"]=61;
    ...
    string city_name;
    cout << "\nEnter a city: ";
    cin >> city_name;
    if (city_num.end()== city_num.find(city_name))
        cout << city_name << " is not in the database" << endl;
    else
        cout << "Number of " << city_name << ": " << city_num[city_name];
}
```

MultiMap

- ▶ `#include <map>`
- ▶ Main difference between map and multimap is that the multimap supports multiple entries of values having the same keys and the same values.

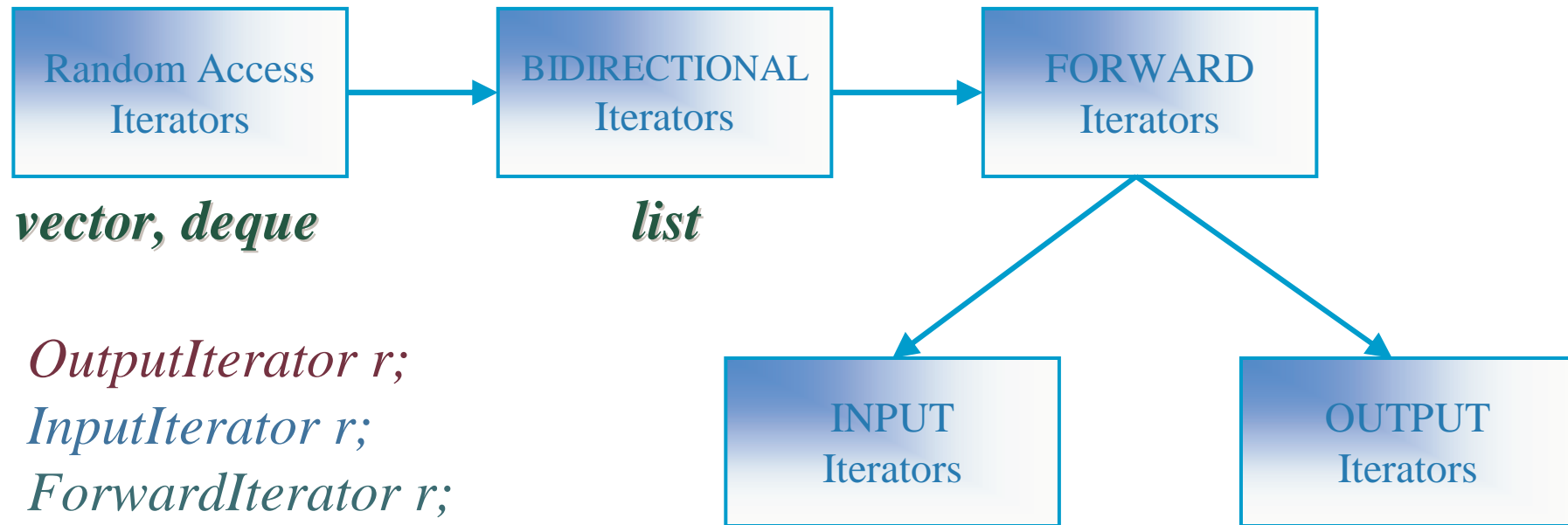
Özetçe

| İşlem | Yürütülen İşlem |
|--------------|---------------------|
| a.size() | a.end() – a.begin() |
| a.max_size() | |
| a.empty() | a.size() == 0 |

| İşlem | Dönüş Değeri | Yürütülen İşlem | Uygulanabildiği Kaplar |
|-----------------|--------------|-----------------------|------------------------|
| a.front() | T& | *a.begin() | vector, list, deque |
| a.back() | T& | *a.end() | vector, list, deque |
| a.push_front(x) | void | a.insert(a.begin(),x) | list,deque |
| a.push_back(x) | void | a.insert(a.end(),x) | vector, list,deque |
| a.pop_front() | void | a.erase(a.begin()) | list,deque |
| a.pop_back() | void | a.erase(--a.end()) | list,deque |
| a[n] | T& | *(a.begin()+n) | vector,deque |

Iterators

Iterators : Genelleştirilmiş İşaretçi



OutputIterator r;
InputIterator r;
ForwardIterator r;
BidirectionalIterator r;
RandomIterator r ;

Iterator Capability

| Iterator Capability | Input | Output | Forward | Bidirectional | Random Access |
|----------------------------|-------|--------|---------|---------------|---------------|
| Dereferencing read | yes | no | yes | yes | yes |
| Dereferencing write | no | yes | yes | yes | yes |
| Fixed and repeatable order | no | no | yes | yes | yes |
| ++i i++ | yes | yes | yes | yes | yes |
| --i i-- | no | no | no | yes | yes |
| i[n] | no | no | no | no | yes |
| i + n | no | no | no | no | yes |
| i - n | no | no | no | no | yes |
| i += n | no | no | no | no | yes |
| i -= n | no | no | no | no | yes |

Output Iterators

① *OutputIterator a ;*

...

**a=t ;*

*t = *a ;* *Hata*

③ *OutputIterator r ;*

...

r++ ;

r++ ; *Hata*

② *OutputIterator r ;*

...

**r=0 ;*

**r=1 ;* *Hata*

④ *OutputIterator i,j ;*

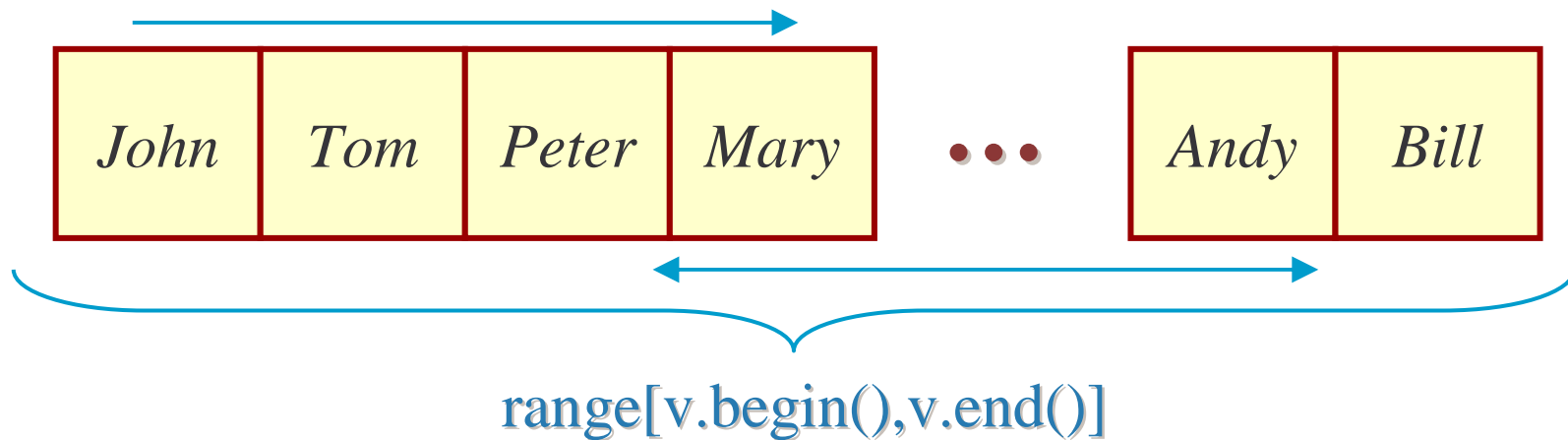
...

i=j ;

**i++=a ;* *Hata*

**j=b ;*

Forward and Bidirectional Iterators



```
list<int> l (1,1) ;  
l.push_back(2) ; // list l : 1 2  
list<int>::iterator first=l.begin() ;  
list<int>::iterator last=l.end() ;  
while( last != first){  
    -- last ;  
    cout << *last << " " ;  
}
```

```
template<class ForwardIterator, class T>
ForwardIterator find_linear(ForwardIterator first,
                          ForwardIterator last, T& value){
    while( first != last) if( *first++ == value) return first;
                          else return last ;
}

vector<int> v(3,1) ;
v.push_back(7); // vector : 1 1 1 7
vector<int>::iterator i=find_linear(v.begin(), v.end(),7) ;
if(i != v.end() ) cout << *i ;
else cout << “not found!” ;
```

Bubble Sort

```
template<class Compare>
void bubble_sort(BidirectionalIterator first,
                BidirectionalIterator last, Compare comp){
    BidirectionalIterator left = first , right = first ;
    right ++ ;
    while( first != last){
        while( right != last ){
            if( comp(*right,*left) )
                iter_swap(left,right) ;
            right++ ;
            left++;
        }
        last -- ;
        left = first ; right = first ;
    }
}
```

```
list<int> l ;
bubble_sort(l.begin(),l.end(),less<int>()) ;
bubble_sort(l.begin(),l.end(),greater<int>()) ;
```

Random Access Iterators

```
vector<int> v(1,1) ;
v.push_back(2) ; v.push_back(3) ; v.push_back(4) ; // v : 1 2 3 4
vector<int>::iterator i=v.begin() ;
vector<int>::iterator j=i+2;
cout << *j << “ ” ;
i += 3 ; cout << *i << “” ;
j = i - 1 ; cout << *j << “” ;
j -= 2 ; cout << *j << “” ;
cout << v[1] << endl ;
(j<i) ? cout << “j < i” : cout << “not j < i” ; cout << endl ;
(j>i) ? cout << “j > i” : cout << “not j > i” ; cout << endl ;
(j>=i) && (j<=i)? cout << “j and i equal” : cout << “j and i not equal > i” ; cout <<
endl ;
i = j ;
j= v.begin();
i = v.end ;
cout << “iterator distance end – begin : ” << (i-j) ;
```


Iterator Operators

- ▶ STL provides two functions that return the number of elements between two elements and that jump from one element to any other element in the container:
 - distance()
 - advance()

distance()

- ▶ The distance() function finds the distance between the current position of two iterators.

```
template<class RandomAccessIterator>
iterator_traits<RandomAccessIterator>::difference_type
distance(RandomAccessIterator first, RandomAccessIterator
    last) {
    return last - first;
}
template<class InputIterator>
iterator_traits<InputIterator>::difference_type
distance(InputIterator first, InputIterator last) {
    iterator_traits<InputIterator>::difference_type n = 0;
    while (first++ != last) ++n;
    return n;
}
```

advance()

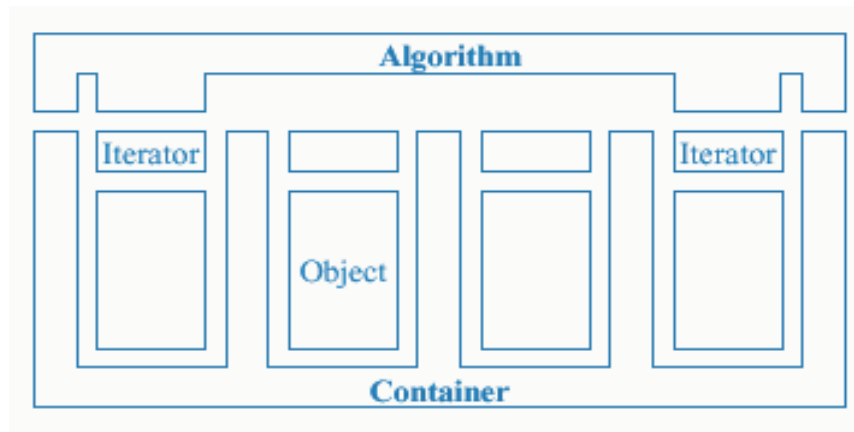
- ▶ So far, we have seen how we can move iterators forward and backward by using the increment and decrement operators, respectively. We can also move random access iterators several steps at a time using the addition and subtraction functions. Other types of iterators, however, do not have the addition and subtraction functions.
- ▶ The STL provides the `advance()` function to move any iterator—except the output iterators—several steps at a time:

```
template<class InputIterator, class Distance>
void advance(InputIterator& ii, Distance& n) {
    while (n-->0) ++ii;
}

template<class BidirectionalIterator, class Distance>
void advance(BidirectionalIterator & bi, Distance& n) {
    if (n >= 0) while (n-->0) ++bi;
    else while (n++<0) --bi;
}

template<class RandomAccessIterator, class Distance>
void advance(RandomAccessIterator& ri, Distance& n) {
    ri += n;
}
```

Designing Generic Algorithm



implementing an algorithm such as computing the maximum value in a sequence can be done without knowing the details of how values are stored in that sequence:

```
template <class Iterator>
```

```
Iterator max_element (Iterator beg, // refers to start of collection  
                    Iterator end) // refers to end of collection
```

```
{
```

```
    ...
```

```
}
```

Binary Search for Integer Array

```
const int * binary_search(const int * array, int n, int x){
    const int *lo = array, *hi = array + n , *mid ;
    while( lo != hi ) {
        mid = lo + (hi-lo)/2 ;
        if( x == *mid ) return mid ;
        if( x < *mid ) hi = mid ;
        else lo = mid + 1 ;
    }
    return 0 ;
}
```

Binary Search—Template Solution (Form-1)

```
template<class T>
const T * binary_search(const T * array, int n, T& x){
    const T *lo = array, *hi = array + n , *mid ;
    while( lo != hi ) {
        mid = lo + (hi-lo)/2 ;
        if( x == *mid ) return mid ;
        if( x < *mid ) hi = mid ;
        else lo = mid + 1 ;
    }
    return 0 ;
}
```

Binary Search—Template Solution (Form-2)

```
template<class T>
const T * binary_search(T * first, T * last, T& x){
    const T *lo = first, *hi = last, *mid ;
    while( lo != hi ) {
        mid = lo + (hi-lo)/2 ;
        if( x == *mid ) return mid ;
        if( x < *mid ) hi = mid ;
        else lo = mid + 1 ;
    }
    return last ;
}
```


Generic Binary Search

```
template<class RandomAccessIterator,class T>
const T * binary_search(RandomAccessIterator first,
                       RandomAccessIterator last, T& value){
    RandomAccessIterator not_found = last, mid ;
    while( lo != hi ) {
        mid = first + (last-first)/2 ;
        if( x == *mid ) return mid ;
        if( x < *mid ) last = mid ;
        else first = mid + 1 ;
    }
    return not_found ;
}
```

STL Algorithms

#include <algorithm>

| Algorithm | Purpose |
|------------|--|
| find | Returns first element equivalent to a specified value |
| count | Counts the number of elements that have a specified value |
| equal | Compares the contents of two containers and returns true if all corresponding elements are equal |
| search | Looks for a sequence of values in one container that correspond with the same sequence in another container |
| copy | Copies a sequence of values from one container to another (or to a different location in the same container) |
| swap | Exchanges a value in one location with a value in another |
| iter_swap | Exchanges a sequence of values in one location with a sequence of values in another location |
| fill | Copies a value into a sequence of locations |
| sort | Sorts the values in a container according to a specified ordering |
| merge | Combines two sorted ranges of elements to make a larger sorted range |
| accumulate | Returns the sum of the elements in a given range |
| for_each | Executes a specified function for each element in the container |

find()

- ▶ The find() algorithm looks for the first element in a container that has a specified value.
- ▶ find() example program shows how this looks when we're trying to find a value in an array of int's.

Example

```
#include <iostream>

#include <algorithm>           //for find()

int arr[] = { 11, 22, 33, 44, 55, 66, 77, 88 };

int main() {

    int* ptr;

    ptr = find(arr, arr+8, 33);    //find first 33

    cout << "First object with value 33 found at offset "
         << (ptr-arr) << endl;

    return 0;

}
```

count()

- ▶ count() counts how many elements in a container have a specified value and returns this number.

```
#include <iostream>
#include <algorithm>    //for count()
int arr[] = { 33, 22, 33, 44, 33, 55, 66, 77 };

int main(){
    int n = count(arr, arr+8, 33); //count number of 33's
    cout << "There are " << n << " 33's in arr." << endl;
    return 0;
}
```

count_if()

- ▶ `size_t count_if(InputIterator first, InputIterator last, Predicate predicate)`

```
#include <vector>
#include <algorithm> //for count_if()
int a[] = { 1, 2, 3, 4, 3, 4, 2, 1, 3 };
class Odd {
public:
    bool operator()(int val){ return val&1 ; }
};
int main(){
    std::vector<int> iv(a,a+9) ;
    std::cout << count_if(iv.begin(),iv.end(),Odd()) ;
    return 0 ;
}
```

equal()

- ▶ `bool equal(InputIterator first, InputIterator last, InputIterator otherFirst)`
- ▶ `bool equal(InputIterator first, InputIterator last, InputIterator otherFirst, Predicate predicate)`

```
class CaseString {
public:
    bool operator()(string const &first,string const &second){
        return !strcasecmp(first.c_str(),second.c_str()) ;
    }
};
int main(){
    string
    first[]={ "Alpha","bravo","Charley","echo","Delta","golf"},
    second[]={ "alpha","Bravo","charley","Echo","delta","Golf"} ;
    std::string *last = first + sizeof(first)/sizeof(std::string) ;
    cout << (equal(first,last,second)?"Equal":"Not equal") ;
    cout << (equal(first,last,second,CaseString())?"Equal":
        "Not equal") ;

    return 0 ;
}
```


fill(), fill_n()

- ▶ void fill(ForwardIterator first, ForwardIterator last,
Type const &value)

```
vector<int> iv(8);  
fill(iv.begin(), iv.end(), 8);
```

- ▶ void fill_n(ForwardIterator first, Size n,
Type const &value)

```
vector<int> iv(8);  
fill_n(iv.begin()+2, 4, 8);
```

sort()

- ▶ You can guess what the sort() algorithm does.

Here's an example:

```
#include <iostream>
#include <algorithm>
int arr[] = {45, 2, 22, -17, 0, -30, 25, 55};
int main(){
    sort(arr, arr+8);           //sort the numbers
    for(int j=0; j<8; j++)      //display sorted array
        cout << arr[j] << ' ';
    return 0;
}
```

search()

- ▶ Some algorithms operate on two containers at once. For instance, while the find() algorithm looks for a specified value in a single container, the search() algorithm looks for a sequence of values, specified by one container, within another container.

```
int source[] = { 11, 44, 33, 11, 22, 33, 11, 22, 44 };
int pattern[] = { 11, 22, 33 };
int main(){
    int* ptr;
    ptr = search(source, source+9, pattern, pattern+3);
    if(ptr == source+9) cout << "No match found\n";
    else                cout << "Match at " << (ptr - source) ;
    return 0;
}
```

binary_search()

▶ #include <algorithm>

- bool binary_search(ForwardIterator first, ForwardIterator last, Type const &value)
- bool binary_search(ForwardIterator first, ForwardIterator last, Type const &value, Comparator comp)

merge()

```
#include <iostream>
#include <algorithm>          //for merge()
using namespace std;
int src1[] = { 2, 3, 4, 6, 8 };
int src2[] = { 1, 3, 5 };
int dest[8];
int main() {                  //merge src1 and src2 into dest
    merge(src1, src1+5, src2, src2+3, dest);
    for (int j=0; j<8; j++)   //display dest
        cout << dest[j] << ' ';
    cout << endl;
    return 0;
}
```

accumulate()

▶ #include <numeric>

- Type `accumulate(InputIterator first, InputIterator last, Type init)`
operator+() is applied to all elements and the result is returned
- Type `accumulate(InputIterator first, InputIterator last, Type init, BinaryOperation op)`
binary operator op() is applied to all elements

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

```
int main(){
    int ia[]={ 1,2,3,4} ;
    std::vector<int> iv(ia,ia+4) ;

    cout << accumulate(iv.begin(),iv.end(),int()) << std::endl ;
    cout << accumulate(iv.begin(),iv.end(),int(1),multiplies<int>())
        << endl ;
    system("pause") ;
    return 0 ;
}
```

adjacent_difference()

▶ #include <numeric>

- OutputIterator adjacent_difference(InputIterator first, InputIterator last, OutputOperator result)
- OutputIterator adjacent_difference(InputIterator first, InputIterator last, OutputOperator result, BinaryOperation op)


```
#include <iostream>
#include <numeric>
#include <vector>
int main(){
    int ia[]={ 1,3,7,23 } ;
    std::vector<int> iv(ia,ia+4) ;
    std::vector<int> ov(iv.size()) ;
    adjacent_difference(iv.begin(),iv.end(),ov.begin()) ;
    copy(ov.begin(),ov.end(),std::ostream_iterator<int>(cout," ")) ;
    std::cout << std::endl ;

    adjacent_difference(iv.begin(),iv.end(),ov.begin(),minus<int>()) ;
    copy(ov.begin(),ov.end(),ostream_iterator<int>(cout," ")) ;
    system("pause") ;
    return 0 ;
}
```

copy(), copy_backward()

▶ #include <algorithm>

- OutputIterator copy(InputIterator first, InputIterator last, OutputIterator destination)
- BidirectionalIterator copy(InputIterator first, InputIterator last, BidirectionalIterator last2)

for_each

- ▶ Function `for_each(ForwardIterator first, ForwardIterator last, Function func)`

```
void lowerCase(char &c){
    c = static_cast<char>(tolower(c)) ;
}
void capitalizedOutput(std::string const &str){
    char *tmp = strcpy(new char[str.size()+1],str.c_str()) ;
    std::for_each(tmp+1,tmp+str.size(),lowerCase) ;

    tmp[0] = toupper(*tmp) ;
    std::cout << tmp << " " ;
    delete []tmp;
}
```



foreach1.cpp

```
int main(){
    std::string
        sarr[] =
        {
            "alpha", "BRAVO", "charley", "ECHO", "delta",
            "FOXTROT", "golf", "HOTEL"
        },
        *last = sarr + sizeof(sarr) / sizeof(std::string) ;
    void (*f)(std::string const&) ;
    f = std::for_each(sarr,last,capitalizedOutput) ;
    std::cout << std::endl ;
    f("alpha") ;
    std::cout << std::endl ;
    system("pause") ;
    return 0 ;
}
```

Another Example

```
class Show{
    int d_count ;
public:
    void operator()(std::string &str){
        for_each(str.begin(),str.end(),lowerCase) ;
        str[0] =toupper(str[0]);
        std::cout << ++d_count << " " << str << "; " ;
    }
    int getCount() const{
        return d_count ;
    }
};
```



```
int main(){
    std::string
        sarr[] = {
            "alpha", "BRAVO", "charley", "ECHO", "delta",
            "FOXTROT", "golf", "HOTEL"
        },
        *last = sarr + sizeof(sarr) / sizeof(std::string) ;
    cout << for_each(sarr,last,Show()).getCount() << endl ;
    system("pause") ;
    return 0 ;
}
```

transform()

```
int _3_n_plus_1(int n) {  
    return (n&1) ? 3*n+1 : n/2 ;  
}
```

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

```
int main(){  
    int iArr[] = { 5,2,23,76,33,44} ;  
    std::for_each(iArr,iArr+6,show) ; std::cout << std::endl ;  
    std::transform(iArr,iArr+6,iArr,_3_n_plus_1) ;  
    std::for_each(iArr,iArr+6,show) ;  
    system("pause") ;  
    return 0 ;  
}
```



transform.cpp

Predicates in <functional>

- ▶ When the type of the return value of a unary function object is `bool`, the function is called a unary predicate. A binary function object that returns a `bool` value is called a binary predicate.
- ▶ The Standard C++ Library defines several common predicates in <functional>

TABLE . PREDICATES DEFINED IN <functional>

| <i>Function</i> | <i>Type</i> | <i>Description</i> |
|----------------------------|-------------|-----------------------------------|
| <code>equal_to</code> | binary | <code>arg1 == arg2</code> |
| <code>not_equal_to</code> | binary | <code>arg1 != arg2</code> |
| <code>greater</code> | binary | <code>arg1 > arg2</code> |
| <code>greater_equal</code> | binary | <code>arg1 >= arg2</code> |
| <code>less</code> | binary | <code>arg1 < arg2</code> |
| <code>less_equal</code> | binary | <code>arg1 <= arg2</code> |
| <code>logical_and</code> | binary | <code>arg1 && arg2</code> |
| <code>logical_or</code> | binary | <code>arg1 arg2</code> |
| <code>logical_not</code> | unary | <code>!arg1</code> |

```
template<class T>
class equal_to : binary_function<T, T, bool> {
bool operator()(T& arg1, T& arg2) const { return arg1 == arg2; }
};
```