**#include <vector>**

vector<T> d1;

vector<T> d2(n);

vector<T> d3(n,t);

vector<T> d4(n,t,a1);

vector<T> d5(it1,it2,a1);

vector<T> d6(it1,it2);

vector<T> v2(v1); Construieste v2 din v1 prin copiere

vector<T> v3(b+1,b+3); Preia un interval din b[](tablou unidimensional);intervalul:[b+1,b+3)

vector<T> v4(b,b+5)

vector<T> v1(&v[0],&v[0]+5); Copiaza in v1 cinci elemente ale lui v

v.size();

v.empty(); returneaza true or false

v.resize(n);

v.resize(n,x);

v.size\_max();

v.push\_back(x);

v.pop\_back();

v.back();

v.front();

v.clear();

v.assign(it1,it2); Inlocuieste secventa controlata

de v, cu secventa cuprinsa in intervalul [it1,it2)

v.assign(n,x); Inlocuieste secventa controlata

de v,cu o secventa formata din n elem de valoare x

**Operatii cu obiecte de tip vector**

-operatorii relationali(toti operatorii relationali compara elementele vectorilor dpv lexicografic

-operatorul de atribuire

v1.swap(v2); Interschimba v1 cu v2(functie membra)

swap(v1,v2); Interschimba v1 cu v2 (functie globala)

**Iteratori**

vector<T>::iterator it1;

vector<T>::const\_iterator it2;

it=v.begin();

**Clasa Template vector<bool>**

vector<bool> v(3); //v=[false,false,false]

v[0]=~v[1]; //v=[true,false,false]

v.flip(); //v=[false,true,true]

vector<bool>::swap(v[0],v[1]); //v=[true,false,true]

it=v.end();

**Reverse iterators**

vector<T>::reverse\_iterator rit1;

vector<T>::const\_reverse\_iterator rit2;

rit1=v.rbegin();

rit2=v.rend();

sau

vector<int>::reverse\_iterator r1(v.end());

vector<int>::reverse\_iterator r2(v.begin());

v.capacity(); returneaza numarul total de elemente pe care vectorul le poate pastra,

fara sa fie necesara realocarea memoriei.

Realocarea invalideaza toate referintele,pointerii si iteratorii pentru elementele unui vector.

v.reserve();

v. ̴vector<T>() destructor

**Idiom-ul "swap trick"** Singura cale sigura de a elibera toata memoria rezervata este folosirea acestui idiom.

vector<T>().swap(v);

Idiomul "shrink-to-fit"

vector<T>(v).swap(v);

**Operatii de inserare in vectori:**

itr=v.insert(it,x); returneaza un iterator care indica noul element inserat

v.insert(it,n,x);

v.insert(it,it1,it2);

**Operatii de stergere in vectori:**

it=v.erase(it);

it=v.erase(it1,it2);

**#include <deque>**

deque<T> q1; Creeaza o coada vida

deque<T> q2(n); Creeaza o coada cu n elemente

deque<T> q3(n,t); Coada cu n elemente de valoare t

deque<T> q4(q3),q5=q4; Creeaza q4 prin copiere din q3 si creeaza q5 prin copiere din q4

int a[]={100,200,300,400};

vector<int> v(a,a+4);

deque<int> q4(&v[0],&v[0]+4);

d.push\_front(x);

d.pop\_front();

d.assign(n,x); insereaza n elem de valoare x;

iter=d.insert(it,x); insereaza un singur element cu valoare x in secventa controlata de d si returneaza un iterator

care indica noul element inserat

d.insert(it,n,x); insereaza n elemente cu valoare x in secventa controlata de d inaintea elementului indicat de it

d.insert(it,it1,it2); insereaza secventa definita de [it1,it2) inaintea elementului indicat de it

iter=d.erase(it) sterge elementul indicat de it din secventa controlata de d si returneaza un iterator care

desemneaza primul element ce ramane dincolo de ultimul element sters,sau end() daca un

asemenea element nu exista

iter=d.erase(it1,it2) sterge elementele din intervalul [it1,it2) din secventa controlata de d;

#include <list>

**Functii care combina elementele listelor**

Imbinarea listelor cu splice()

L.splice(it,x); insereaza lista controlata de x, inaintea elementului indicat de it, in lista controlata de obiectul L.

De asemenea,inlatura toate elementele din x. Cele doua liste trebuie sa fie diferite.

L.splice(it,x,it1); inlatura elementul indicat de it1 din secventa controlata de x si il insereaza inaintea elementului

indicat de it in lista L.Daca it==it1 sau it==++it1,nu se executa nimic

L.splice(it,x,it1,it2); insereaza intervalul desemnat de [it1,it2) in lista controlata de x,inaintea elementului indicat de

it in secventa controlata de L.Daca &x==this (adica L si X coincid),atunci intervalul [it1,it2) nu

trebuie sa includa elementul indicat de it

Interclasarea listelor cu merge()

L.merge(x); Cele doua functii membre merge imbina listele L si x prin inlaturarea tuturor elementelor din

L.merge(x,pred); lista x si inserarea acestora in lista L.Cele doua liste trebuie sa fie ordonate,iar ordonarea sa fie

facuta dupa acelasi predicat.Secventa care rezulta este de asemenea ordonata dupa acelasi

predicat.

ex: L1.merge(L2,greater<int>()); ///Ordoneaza descrescator

**Functii care inlatura elemente din liste**

L.unique(); inlatura din lista L toate elementele care "compara egal" cu elementul precedent.Altfel spus elimina

toate elementele in afara de primul din fiecare grup consecutiv de elemente cu ordine echivalenta.

L.unique(pr); inlatura din lista L toate elementele in afara de primul,din fiecare grup consecutiv

de elemente cu "ordine echivalenta"

L.remove(x); inlatura din lista L toate elementele desemnate de iteratorul it, pentru care \*it==x

(Sterge din lista toate elementele de valoare x)

L.remove\_if(pr); inlatura din lista L toate elementele desemnate de iteratorul it, pentru care pr(\*it) este true

**Sortarea si inversarea listelor**

Daca it1 si it2 sunt doi iteratori corespunzatori pozitiilor i si j in lista,atunci:

L.sort(); impune ordinea: !(\*it2<\*it1) pentru oricare i si j,cu i<j;

L.sort(pr); impune ordinea: !pr(\*it2,\*it1) pentru oricare pozitii i si j,cu i<j;

L.reverse() inverseaza ordinea in care elementele apar in secventa.

Operatia este realizata intern,printr-o succesiune de operatii splice.

Cele doua functii sort() nu invalideaza niciun iterator.Iteratorii raman valizi si continua sa indice aceleasi elemente.

**#include <string>**

**Construirea stringurilor**

1.string() Construieste un string vid

2.string(s) Construieste un string prin copiere

string s1(s); din obiectul s de tip string

3.string(s,p) Construieste un string prin copierea

string s1(s,p); elementelor din stringul s,incepand cu pozitia p

4.string(s,p,n) Construieste un string prin copierea a n elemente

string s1(s,p,n); din stringul s,incepand cu pozitia p

5.string(n,c) Construieste un string cu n elemente de

string s1(n,c); valoare c fiecare

6.string(a) Construieste un string prin copierea elementelor

string s(a); din a (a este un sir standard C)

7.string(a,n) Construieste un string cu n elemente prin

string s(a,n); copierea elementelor din a,care este string C

8.string(it1,it2) Construieste un string prin copierea

string s(it1,it2); elementelor din intervalul [it1, it2);

string s="VOIOS!";

string s("abcd");

string s(&a[0], &a[18]);

s.length(); returneaza dimensiunea stringului,adica numarul de caratere

s.size(); gestionat de obiectul de tip string;

///s[11]='E'; Comportament nedefinit

try{

s.at(11)='E'; ///Arunca exceptia out\_of\_range

}catch(exception e){

cout<<"Exceptie!";

}

**Functie de comparatie:(compara lexicografic) compare()**

s1.compare(s2); compara s1 si s2 din punct de vedere lexicografic daca s1<s2,returneaza <0,...

s1.compare(p1,n1,s2,p2,n2); compara subsirul din s1 care incepe la pozitia p1 si are lungimea n1,cu subsirul din s2

care incepe la pozitia p2 si are lungimea n2;

s1.comapare(p1,n1,a,p2,n2) compara subsirul din s1 care incepe la pozitia p1 si are lungimea n1,cu subsirul din

stringul C cu numele a,care incepe la pozitia p2 si are lungimea n2;

**Operatori relationali:**

>,>=,<,<=,==,!=

Comparatiile se fac din punct de vedere lexicografic,caracter cu caracter, de la inceputul catre sfarsitul secventei.

string s1("MAR");

a[]="PAR";

s1 < a // operatiile binare relationale admit ca unul dintre operanzi sa fie string C

**Atribuiri:**

s1.assign(s2); atribuie lui s1 o copie a stringului s2

s1.assign(s2,p,n); atribuie obiectului s1 n caractere copiate din s2 incepand cu pozitia p;

s1.assign("DOISPREZECE",p,n); atribuie obiectului s1 n caractere copiate din stringul C "DOISPREZECE",

incepand cu pozitia p

s1.assign(n,'w'); atribuie obiectului s1 n caractere 'W'

s1.assign("ABCDE",n); atribuie obiectului s1 primele n caractere din "ABCDE"

string::npos returneaza numarul maxim de caractere pe care il puteti stoca intr-un string

sintaxa de accesare a membrilor statici in C++:nume\_clasa::nume\_membru\_static

**Operatii de concatenare**

operatorul +;

string + string

string + string C

string C + string

string + caracter

s1.append(a); concateneaza stringul C a la sfarsitul lui s1

s1.append(a,p,n); concateneaza n elemente din stringul C a la sfasitul lui s1,incepand de la pozitia p

s2.append(s1); concateneaza stringul s1(s2+=s1) la sfarsitul lui s2

s2+=' '; si s2+="I'm here"; concateneaza un caracter,respectiv un string C

s2.append(s3,p,n); concateneaza n elemente din s3 la sfarsitul lui s2,incepand de la pozitia p;

s2.append(v.begin()+1,v.begin()+3); concateneaza intervalul [v.begin(),v.begin()+3) din vectorul v la sfarsitul lui s2.

cout<<s2.append(n,'.'); concateneaza n caractere '.';

cout<<s1.append("in zone",p,n);

s2.push\_back(' ');

**Conversii intre stringuri si siruri standard**

s.c\_str(); creeaza un sir de caractere terminat cu '\0',avand toate caracterele copiate din s si returneaza un pointer

const char\* care indica adresa acestuia

**Operatii de inserare:**

cout<<s1.insert(p,s2); insereaza s2 in s1 la pozitia (indexul) p

cout<<s1.insert(p1,s2,p2,n2); insereaza un substring al lui s2 in s1 la pozitia p1; Substringul incepe in s2 la

pozitia p2 si are lungimea n2;

cout<<s1.insert(p,"WWW"); insereaza stringul C "WWW" in s1 la pozitia p;

cout<<s1.insert(p,n,'K'); insereaza n caractere 'K' in s1,la pozitia p;

s1.insert(s1.end(),'T'); insereaza caracterul 'T' imediat inainte de pozitia indicata de iteratorul

s1.end()(inserare la sfarsit)

s1.insert(s1.end(),v.begin(),v.end()); insereaza secventa [v.begin(),v.end()) din vectorul v imediat inaintea

pozitiei indicate de iteratorul s1.end() (inserare la sfarsit)

**Operatii de stergere:**

it=s.erase(begin()+1); sterge elementul indicat de s.begin()+1;

it=s.erase(s.begin()+1,s.begin()+3); sterge intervalul definit de [s.begin()+1,s.begin()+3)

s.erase(p); sterge elementul de pe pozitia p;

s.erase(p); si s.erase(); sunt apeluri ale functiei s.erase(pos=0,n=npos) care sterge n

caractere,incepand cu pozitia pos.

**Operatii de inlocuire:**

cout<<s1.replace(p,n,s2); inlocuieste in s1 incepand cu pozitia p,un numar de n

elemente, cu primele n caractere din s2.Daca s2.size()>n,

caracterele urmatoare din s2 nu suprascriu ci se insereaza

cout<<s1.replace(p1,n1,s2,p2,n2); inlocuieste in s1,n1 caractere,incepand cu pozitia

p1,cu n2 caractere din s2,care incep la pozitia p2

cout<<s1.replace(s1.begin(),s1.begin()+1,s2);

inlocuieste caracterele din intervalul [s1.begin(),s1.begin()+1)

cu o copie a tuturor caracterelor din s2

cout<<s1.replace(s1.begin(),s1.begin()+8,s2.begin(),s2.end());

cout<<s1.replace(i,n1,n2,c); inlocuieste n1 elemente din stringul curent,incepand cu pozitia

i,cu n2 elemente de valoare c.

cout<<s1.replace(it1,it2,s,n); inlocuieste intervalul specificat de [it1,it2) cu primele

n elemente din stringul C,s

/\

|| nu e necesar sa folosesti cout:

ex:s1.replace(1,3,s2);

**Functii de cautare:**

1 size\_type find(const string& s,size\_type i=0) const

returneaza indicele primei aparitii a stringului s in stringul apelant.

Cautarea incepe de indexul i.returneaza string::npos daca nu este gasit

2 size\_type find(const char\* s,size\_type i=0) const

returneaza indicele primei aparitii a sirului s in stringul apelant.

Cautarea incepe la indexul i.Returneaza string::npos daca nu este gasit

3 size\_type find(const char\* s,size\_type i=0,size\_type n=0) const

Returneaza indicele primei aparitii a primelor n caractere ale sirului s

in stringul apelant.Cautarea incepe la indexul i.Returneaza string::npos

daca nu este gasit

4 size\_type find(char c,size\_type i=0) const

Returneaza indicele primei aparitii a caracterului c in stringul apelant.

Cautarea incepe la indexul i.Returneaza string::npos daca nu este gasit.

5 size\_type find\_first\_of(const string& s,size\_type i=0) const

Returneaza indicele primului caracter din stringul apelant,care concide cu

oricare caracter din s.Cauatrea incepe la indexul i.Returneaza string::npos

daca niciun caracter nu este gasit

6 size\_type find\_first\_of(const char\* s,size\_type i=0) const

Returneaza indicele primului caracter din stringul apelant,care concide cu

oricare caracter din sirul s.Cautarea incepe la indexul i.Returneaza string::npos

daca niciun caracter nu este gasit

7 string substr(const size\_type i=0,size\_type n=npos) const

Returneaza substringul de lungime n,care incepe la pozitia i in stringul apelant

s1.rfind(s2); ,s1.rfind(s2,2); ,s1.rfind("CDE"); ,s1.rfind("CDE",2);,

returneaza indicele ultimei aparitii in stringul apelant al substringului cautat

#include <iostream>

#include <string>

using namespace std;

void Scrie(string::size\_type i)

{

if(i!=string::npos)

cout<<"Gasit!Incepe la pozitia"<<i<<'\n';

else

cout<<"Nu s-a gasit\n";

}

int main()

{

string s1("ABCDEF"), s2;

string::size\_type pos;

cin>>s2;

pos =s1.find(s2); //1

Scrie(pos);

pos =s1.find(s2,2); //2

Scrie(pos);

pos =s1.find("CDE"); //3

Scrie(pos);

pos =s1.find("CDE",2); //4

Scrie(pos);

pos =s1.find\_first\_of("KDAT"); //6

Scrie(pos);

cout<<s1.substr(2,3); //7

pos =s1.rfind(s2);

Scrie(pos);

pos =s1.rfind(s2,2);

Scrie(pos);

pos =s1.rfind("CDE");

Scrie(pos);

pos =s1.rfind("CDE",2);

return 0}

**#include <queue>**

Exemple de declarare

queue<int> q1; coada vida de intregi

queue<string, list<string> > q2 coada vida de stringuri care foloseste un obiect de tip list<string> in loc de

deque<string> in calitate de container intern

push() insereaza un element in coada(il adauga la sfarsitul cozii);

pop() inlatura un element din coada(cel care a fost inserat primul);

back() returneaza ultimul element din coada(cel care a fost inserat ultimul);

front() returneaza primul element din coada(cel care a fost inserat primul);

empty() returneaza true atunci cand coada este vida;

size() returneaza numarul de elemente din coada;

queue<T> nu defineste iteratori

**#include <queue>**

priority\_queue<int> q1; coadă vidă de int

priority\_queue<char,deque<char> > q2 coadă vidă de elemente de tip char,care foloseşte ca şi container

intern deque

priority\_queue<string, deque<string>, greater<string> > q3;

coada vida de stringuri,care foloseste ca si container intern un obiect

deque<string> in locul unui vector<string> si greater<string>,ca si

criteriu de comparatie(elementul cu prioritatea minima va fi in capul cozii)

priority\_queue() creeaza o coada vida;

priority\_queue(it1,it2) creeaza o coada in care se copiaza toate elementele din

intervalul [it1,it2),unde it1 si it2 sunt cel puțin iteratori de intrare

push() inserează elementul x in coada;

pop() înlătura un element din coada(cel cu prioritatea maxima);

top() returnează primul element din coada(cel cu prioritatea maxima);

empty() returnează true atunci când coada este vida

size() returnează numărul de elemente din coada

priority\_queue nu definește operatori relationali si nici iteratori

/// Max-Heap

priority\_queue<int,vector<int>,greater<int> > q1;

/// Min-Heap

priority\_queue<int> q;

**#include <stack>**

Exemple de declarare:

stack<int> st1; stiva vida de intregi

stack<char,vector<char> > st2; stiva vida de caractere,care foloseste vector<char>

in loc de deque<char> ca si container intern

s.push(x); insereaza o copie a lui x,care ca fi primul element in stiva

s.pop(); extrage elementul din varful stivei

s.top(); returneaza referinte la elementul din varful stivei(acesta este ultimul adaugat)

s.empty();

s.size();

stack<T> nu defineste iteratori

**#include <set>**

set<int> s; declara un set vid de elementede tip int.

Elementele care se vor insera ulterior in set,se ordoneaza in mod

automat crescator

set<double, greater<double> > s; declara un set vid de double.

Elementele vor fi ordonate descrescator.

int a[]={45,34,23,26};

set<int> s(a,a+4); declara un set de int in care se copiaza elementele din sirul a[].

Elementele vor fi ordonate crescator.

char a[]="CAB";

set<char,greater<char> > s(a,a+3); declara un set de char,in care se insereaza elemente

copiate din sirul a[].Elementele vor fi ordonate

descrescator (s={'C','B','A'})

**Iteratorii claselor set si multiset:**

iterator begin(); returneaza un iterator bidirectional la primul

const\_iterator begin() const; element al set-ului

iterator end(); returneaza un iterator bidirectional care

const\_iterator end() const indica dincolo de ultimul element al set-ului.

reverse\_iterator rbegin(); returneaza un reverse iterator bidirectional care

const\_reverse\_iterator rbegin() const; indica imediat dincolo de sfasitul secventei.

Astfel desemneaza inceputul seventei inversate.

reverse\_iterator rend();

const\_reverse\_iterator rend() const; returneaza un reverse iterator bidirectional care

indica primul element al secventei.Astfel,

desemneaza sfarsitul secventei inversate

**Operatii care nu schimba containerul**

Functiile membre de informare:

empty();

size();

max\_size();

Functii membre de observare:

key\_comp();

value\_comp();

OBS: Pentru set si multiset cheia coincide cu valoarea. Prin urmare,key\_comp() si value\_comp() lucreaza identic.

**Functii membru:**

s.insert(x); insereaza un element de valoare x in containerul s.

Pentru set: Functia determina daca in container mai exista un element y care are ordine

echivalenta cu cea a lui x.Daca nu,atunci creeaza un asemenea element y si il initializeaza

cu x,respectiv creeaza un iterator it care-l indica pe y.Daca inserarea are loc functia returneaza

perechea pair(it,true).Altfel,returneaza pair(it,false).Mai putin riguros,daca elementul x exista deja

in set,inserarea nu mai are loc si membrul bool al perechii indica esuarea prin valoarea false.

Pentru multiset.Inserarea reuseste intotdeauna deoarece multiset admite elemente

cu ordine echivalenta.Functia insereaza elementul x in container,apoi returneaza un

iterator care desemna elementul inserat.

s.insert(it,x); insereaza un element de valoare x in containerul s,folosind it ca punct de plecare pentru cautarea

pozitiei de inserare. Pentru set nu se insereaza un element duplicat.Functia returneaza

Functia returneaza un iterator la elementul de valoare x nou inserat sau care exista deja.

s.insert(it1,it2); insereaza in s o copie a tuturor elementelor din intervalul [it1,it2).

s.erase(it); sterge din s elementul indicat de it

s.erase(x); sterge din s toate elementele care au cheia egala cu x.Returneaza numarul de elemente

sterse(maxim 1 pentru set).

s.erase(it1,it2); sterge din s toate elementele din intervalul [it1,it2)

s1.swap(s2); interschimba continutul containerelor s1 si s2.Containerele trebuie sa aiba amandoua acelasi

tip:doua set-uri sau doua multiset-uri.

s.clear(); sterge toate elementele din container

Operatiile de inserare si stergere nu invalideaza iteratorii care indicau elemente in set-uri si multiset-uri, cu exceptia acelor iteratori care indicau elementele sterse.

typedef set<int>::iterator IS;

pair<IS,bool> p; ///p-obiect de tip pereche

if(p.second==true)

cout<<\*p.first;

pair<T1,T2> p;

p are doi membri:

p.first de tip T1;

p.second de tip T2;

**Operatii speciale de cautare:**

s.find(x); cauta in containerul s primul element a carui cheie are ordine de sortare echivalenta

cu x si returneaza un iterator la elementul gasit.Daca un asemenea element nu exista,

atunci returneaza end();

s.count(x); returneaza numarul de elemente cu cheia echivalenta cu x din containerul s

(pentru set,0 sau 1)

s.lower\_bound(x); returneaza un iterator la primul element a carui cheie nu este mai mica decat x.

Daca un asemenea element nu exista functia returneaza end();

s.upper\_bound(x); retuneaza un iterator la primul element a carui cheie este mai mare decat x.Daca

un asemenea element nu exista,functia returneaza end();

s.equal\_range(x) gaseste un interval care contine toate elementele a caror cheie este x.

Returneaza o pereche p de iteratori,astfel incat p.first==lower\_bound(x)

si p.second==upper\_bound(x).

"\*equal\_range('b'):";

for(it=ms.equal\_range('b').first; ;++it)

{

cout<<\*it<<' ';

if(it==ms.equal\_range('b').second) break;

}

Functiile membre de cautare sunt mai rapide decat algoritmii STL cu acelasi nume. De aceea,trebuie preferate intotdeauna.

**#include <map>**

map<int, char> m; construieste un map vid cu elemente de tip pair<int,char>, utilizand

constructorul map<int,char>().Cheia este de tip int si valoarea de tip char.

Elementele care vor fi inserate ulterior in map se vor ordona crescator

in mod automat,in raport cu cheia.

map<int, double, greater<int> > m; declara un map vid.Elementele sunt perechi cu cheia int si

valoarea double.Vor fi ordonate descrescator dupa valoarea cheii.

map<int, char> m(it1,it2);

Creeaza un sir de obiecte de tip pereche:

pair<int,char> p[]={ make\_pair(2,'A') ,

make\_pair(3,'B') ,

make\_pair(1,'C') };

Construieste un map cu numele m.Insereaza in acesta copii ale elementelor din intervalul [p,p+3).

map<int,char> m(p,p+3);

**Functii membru:**

m.insert(x); insereaza un element de valoare x in containerul m.

Pentru map.Functia determina daca in container mai exista un element y care are ordine

echivalenta cu cea a lui x.Daca nu,atunci creeaza un asemenea element y si il

initializeaza cu x,respectiv creeaza un iterator it care-l indica pe y.Daca inserarea

are loc functia returneaza perechea pair(it,true).Altfel,returneaza pair(it,false).

Mai putin riguros,daca elementul x exista deja in map,inserarea nu mai are loc si

membrul bool al perechii indica esuarea prin valoarea false.

Pentru multimap.Inserarea reuseste intotdeauna deoarece multimap admite elemente

cu ordine echivalenta.Functia insereaza elementul x in container,apoi returneaza un

iterator care desemneaza elementul inserat.

m.insert(it,x); insereaza un element de valoare x in containerul m,folosind it ca punct de plecare pentru

cautarea pozitiei de inserare. Pentru map nu se insereaza un element duplicat.

Functia returneaza un iterator la elementul de valoare x nou inserat sau care exista deja.

m.insert(it1,it2); insereaza in m o copie a tuturor elementelor din intervalul [it1,it2).

m.erase(it); sterge din m elementul indicat de it

m.erase(x); sterge din m toate elementele care au cheia egala cu x.Returneaza numarul

de elemente sterse(maxim 1 pentru map).

m.erase(it1,it2); sterge din m toate elementele din intervalul [it1,it2)

m1.swap(m2); interschimba continutul containerelor m1 si m2.Containerele trebuie sa aiba amandoua acelasi

tip:doua map-uri sau doua multimap-uri.

m.clear(); sterge toate elementele din container

template <class Container>

void Scrie(Container c)

{

typename Container::iterator it;

for(it=c.begin();it!=c.end();++it)

cout<<it->first<<' '<<it->second<<'\n';

///sau cout<<(\*it).first<<' '<<(\*it).second<<'\n';

cout<<'\n';

}

m.insert(pair<nume, varsta>("Nelu",17));

m.insert(make\_pair("Adi",18));

**Operatorul de indexare specific map.**

map<string,int> agenda;

agenda["Voicu Alin"]=234124; insereaza in map o pereche(string,int) cu cheia "Voicu Alin" si valoarea 234124

agenda["Vladescu Carmen"]=424234;

agenda["Muresan Flavia"]=345345;

Citeste cateva nume si cauta numerele lor de telefon:

string nume;

while( getline(cin,nume) )

if(agenda.find(nume)!=agenda.end())

cout<<"Telefonul lui "<<nume<<" este "<<agenda[nume]<<'\n';

else

cout<<nume<<" nu este in agenda\n";

return 0;

O operatie ca aceasta:agenda["Muresan Flavia"] cauta elementul cu cheia egala cu "Muresan Flavia" si returneaza valoarea asociata,345345.

Daca scrieti cout<<agenda["Adi"] se cauta un element cu cheia "Adi",dar nu se gaseste.Prin urmare,se insereaza un nou element in map,cu cheia "Adi" si valoare implicita int(),adica 0.

Daca nu doriti inserari automate in map, folositi pentru cautare functia membru find() si nu functia operator[].

Tablou bidimensional:

map<int,map<int,int> > M; Cheia are tipul int.Valoarea asociata este un map<int,int>

cin>>M[i][j];

cout<<M[i][j];

**Operatii de cautare:**

m.find(x); cauta in containerul m primul element a carui cheie are ordine de sortare echivalenta

cu x si returneaza un iterator la elementul gasit.Daca un asemenea element nu exista,

atunci returneaza end();

m.count(x); returneaza numarul de elemente din containerul m cu cheia echivalenta cu x,

din intervalul [lower\_bound(x),upper\_bound(x))(pentru map,0 sau 1)

m.lower\_bound(x); returneaza un iterator la primul element a carui cheie nu este mai mica decat x.

Daca un asemenea element nu exista functia returneaza end();

m.upper\_bound(x); retuneaza un iterator la primul element a carui cheie este mai mare decat x.Daca

un asemenea element nu exista,functia returneaza end();

m.equal\_range(x) gaseste un interval care contine toate elementele a caror cheie este x.

Returneaza o pereche p de iteratori,astfel incat p.first==lower\_bound(x)

si p.second==upper\_bound(x).

cout<<"equal\_range(3): ";

it=m.equal\_range(3).first;

do

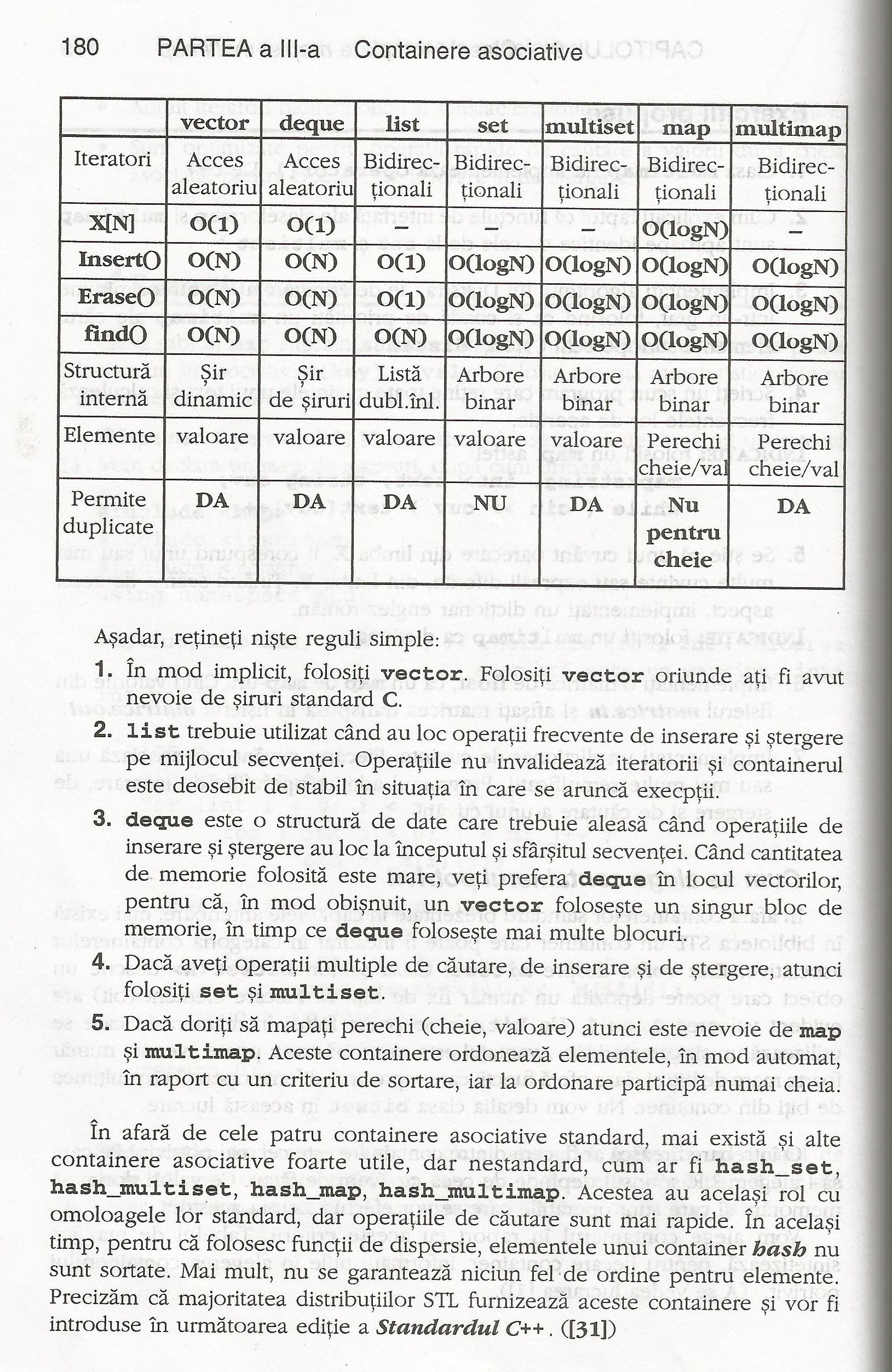
{

cout<<it->first<<' '<<it->second<<' ';

it++;

}

while(it!=m.equal\_range(3).second);



**#include <iterator>**

**Operatiile iteratorilor**

Categorie Proprietati Operatii

Input Citeste.Deplasare numai ==, !=, ++, \*it(doar ca right-value)

inainte (\*it).f,it->f

Output Scrie.Deplasare numai ++,\*it(ca left-value)

inainte

Forward Citeste si scrie numai ==,!=,++,\*it,\*it++,(\*it).f,it->f

inainte

Bidirectional Citeste si scrie inainte ==,!=,++,--,\*it,\*it--,\*it++,(\*it).f,it->f

si inapoi

Random access Citeste si scrie cu acces ==,!=,++,--,\*it,\*it--,\*it++,(\*it).f,it->f,

aleatoriu +,+=,-,-=,it[n],<,<=,>,>=

Cu f s-a notat un membru al clasei container.

**Iteratorii adaptori**

Cauta valoarea 3 si returneaza in it pozitia sa:

deque<int>::iterator it=find(q.begin(),q.end(),3);

Converteste it la reverse\_iterator:

deque<int>::reverse\_iterator rit(it); //Constructor

cout<<'\n'<<\*rit<<'\n';

Converteste rit la iterator:

it=rit.base();

cout<<\*it;

**Back insert iterators:**

int a[]={1,2,3};

list<int> L; //lista vida

list<int>::iterator i;

//Construieste obiectul it de tip back\_insert\_iterator:

back\_insert\_iterator<list<int> > it(L);

\*it=10; //Apeleaza list<int>::push\_back(10);

it++; //Fara efect

\*it=20; //Apeleaza list<int>::push\_back(20);

++it; //Fara efect

it=30; //Acelasi efect cu \*it=30

//Insereaza valorile 1,2,3, folosind iteratorul it

copy(a,a+3,it);

for(i=L.begin();i!=L.end();++i)

cout<<\*i<<' ';

//Insereaza la sfarsit 1 2 3,prin apelul constructorului:

copy(a,a+3,back\_insert\_iterator<vector<int> >(V));

//Insereaza la sfarsit 1 2 3,cu functia back\_inserter():

copy(a,a+3,back\_inserter(V));

Operatiile ++it,it++ nu au alt efect,decat acela ca returneaza iteratorul curent.

Deci it sau \*it reprezinta acelasi lucru:obiectul \*this.

Se pot folosi numai cu containere care implementeaza push\_back()cum ar fi:vector,string,deque,list

**Front insert iterators**

Se pot folosi numai cu containere care implementeaza push\_front().

Singurele containere standard care furnizeaza aceasta functie de interfata sunt list si deque.

int a[]={1,2,3};

deque<int> Q;

deque<int>::iterator it;

//Insereaza la inceput 1 2 3,prin apelul constructorului

copy(a,a+3,front\_insert\_iterator<deque<int> >(Q) );

///Insereaza la inceput 1 2 3, cu funtia front\_inserter()

copy(a,a+3,front\_inserter(Q) );

for(it=Q.begin();it!=Q.end();++it)

cout<<\*it<<' ';

------------------------------------------------------------------------------

**General insert iterators:**

int a[]={1, 2, 3};

vector<int> V;

vector<int>::iterator i;

//Constructorul creeaza un general insert iterator, it

insert\_iterator<vector<int> > it(V, V.end());

\*it=10; //Insereaza 10 inainte de sfarsit

it=20; //Acelasi efect

//inserter() insereaza 1,2,3, dupa al doilea element

copy(a, a+3, inserter(V,V.begin()+1));

for(i=V.begin();i!=V.end();++i)

cout<<\*i<<' ';

Toate containerele standard implementeaza functia insert(it,val).

Prin urmare, un general inserter poate fi utilizat pentru oricare container.

General inserters insereaza un element intr-un container, la o pozitie specificata.

----------------------------------------------------------------------------------

Pentru a face mai comoda crearea acestor iteratori s-au definit functiile nemembre:

back\_inserter(c); care creeaza un back insert iterator pentru containerul c;

front\_inserter(c); care insereaza un element la inceputul containerului c;

inserter(cont, it); containerul cont si it pozitia de inserare

**Caracteristicile insertorilor:**

Nume Clasa Functia apelata Functie de creare

Back inserter back\_insert\_iterator push\_back(val) back\_inserter(cont)

Front inserter front\_insert\_iterator push\_front(val) front\_inserter(cont)

General inserter insert\_iterator insert(iter,val) inserter(cont,iter)

cont - containerul

val - valoarea

iter - pozitia de inserare

**Stream iteratori**

**Ostream iteratori**

Primul constructor:

ostream\_iterator<T>(stream\_iesire); creeaza un obiect de tip ostream\_iterator pentru streamul de

iesire primit ca argument

Exemple:

ostream\_iterator<int>(cout); construieste un obiect anonim de tip ostream\_iterator, care

va insera valori de tip int in stream-ul cout,fara delimitator intre ele

fstream fout("text.out");

ostream\_iterator<char> os(fout); construieste un obiect de tip ostream\_iterator,cu numele os,care

va insera in stream-ul fout elemente de tip char,fara delimitator intre ele

Al doilea constructor:

ostream\_iterator<T>(stream\_iesire, delimitator);

creeaza un obiect de tip ostream\_iterator pentru stream-ul de iesire primit ca

argument,avand ca delimitator intre valorile inserate, sirul de caractere delimitator(de

tip const char\*).

Exemple:

fstream fout("sir.out");

ostream\_iterator<int>(fout," "); construieste un obiect anonim de tip ostream\_iterator,care

va insera in streamu-ul fout valori de tip int,delimitate

intre ele de sirul de caractere " ".

ostream\_iterator<float> os(cout,"\n"); construieste un obiect de tip ostream\_iterator,cu numele os,

care va insera in stream-ul cout,elemente de tip float,

delimitate intre ele de sirul de caractere "\n".

Operatiile \*it, it++, ++it nu fac altceva decat sa returneze it.

Creeaza un ostream iterator pentru stream-ul cout:

ostream\_iterator<char> o1(cout);

Operatii specifice iteratorilor:

\*o1='A'; Insereaza 'A' in stream-ul cout

o1++; Fara efect

o1='B'; Scrie 'B' pe ecran

++o1; Fara efect

o1='C'; Scrie 'C' pe ecran

o1='\n'; Scrie '\n' pe ecran

**Istream iteratori**

Primul constructor:

istream\_iterator<T>(); creeaza un end\_of\_stream iterator

Al doilea constructor:

istream\_iterator<T>(stream\_intrare); creeaza un istream iterator pentru stream-ul de intrare

primit ca argument,din care citeste prima valoare.

Operatia \*it returneaza valoarea citita anterior.

Operatia ++it citeste o valoare si returneaza pozitia acesteia.

Operatia it++ citeste o valoare si returneaza pozitia valorii precedente.

Exemple:

copy(istream\_iterator<string>(cin), //Citeste din cin

istream\_iterator<string>(), //pana la end-of-stream

ostream\_iterator<string>(cout,"\n")); //Scrie in cout

----------------------------------

//Construieste un istream iterator si un end\_of\_stream iterator

istream\_iterator<int>is\_it(cin), eos\_it;

while(is\_it!=eos\_it)

{ //\*is\_it returneaza

cout<<\*is\_it<<" "; //valoarea citita.

++is\_it; //Citeste valoare si

} //returneaza un iterator la ea

**Functori**

**#include <functional>**

Functorii sunt mai rapizi decat pointerii la functii.

Functori predefiniti:

Constructor Apel Efect

plus<T>() plus<T>()(x,y) x + y

minus<T>() minus<T>()(x,y) x - y

multiplies<T>() multiplies<T>()(x,y) x \* y

divides<T>() divides<T>()(x,y) x / y

modulus<T>() modulus<T>()(x,y) x % y

equal\_to<T>() equal\_to<T>()(x,y) x == y

not\_equal\_to<T>() not\_equal\_to<T>()(x,y) x != y

negate<T>() negate<T>()(x) -x

less<T>() less<T>()(x,y) x < y

greater<T>() greater<T>()(x,y) x > y

less\_equal<T>() less\_equal<T>()(x,y) x <= y

greater\_equal<T>() greater\_equal<T>()(x,y) x >= y

logical\_and<T>() logical\_and<T>()(x,y) x && y

logical\_or<T>() logical\_or<T>()(x,y) x || y

logical\_not<T>() logical\_not<T>()(x) !x

int x=4,y=2;

if(greater<int>()(x,y))

cout<<" x > y ";

multiplies<int> M; ///Construieste un obiect functie

cout<<M(x,y);

**Functori adaptori:**

**Adaptorul binder1st**

binder1st este o clasa template de functori,care transforma o functie binara intr-o functie unara.

Primeste doua argumente:un obiect functie func cu doua argumente si o valoare val,si returneaza

un obiect functie cu un singur argument,construit din func,cu primul argument legat la val.

In practica,pentru simplitate,in locul constructorului clasei se prefera utilizarea functiei

nemembre binder1st.

in loc de binder1st<less<int> >(less<int>(),5) scrieti mai simplu bind1st(less<int>(),5).

**Adaptorul binder2nd**

binder2nd este o clasa template de functori,care,la fel ca binder1st transforma o

functie binara intr-o functie unara.Primeste doua argumente:un obiect functie func cu doua

argumente si o valoare val,si returneaza un obiect functie cu un singur argument,construit din

func,cu al doilea argument legat la val.In practica,se prefera utilizarea functiei nemembre

bind2nd.

in loc de binder2nd<greater<int> >(greater<int>(),6) scrieti mai simplu bind2nd(greater<int>,6)

**Adaptori pentru pointeri la functii nemembre**

**ptr\_fun()** primeste ca argument un pointer la o functie si returneaza un functor.Are doua versiuni

supraincarcate:prima adapteaza un pointer la o functie unara,iar cea de a doua adapteaza un

pointer la o functie binara.

Negatorii **not1** si **not2** primesc un predicat unar,respectiv binar si returneaza un functor care este

complementul predicatului original. Negatorii nu admit alte argumente decat functori si nu admit

oricare tip de functori,ci numai functori care mostenesc clasa unary\_function.Din acest motiv,

o incercare de tipul acesta,nu va compila: find\_if(a, a+5, not1(Pred));

eX:

template <class T>

bool Pred(T a,T b) //Predicat binar

{

return a<b;

}

int a[]={20,10,30,10,40};

sort(a,a+5,not2(ptr\_fun(Pred<int>)));

ptr\_fun() primeste un pointer la functia binara Pred<int>() si returneaza un predicat binar.Acesta,

la randul lui,este negat de catre not2().Se realizeaza o sortare descrescatoare,prin adaptarea

predicatului Pred().

**Adaptori pentru pointeri la functii membre**

mem\_fun\_ref() are doua versiuni supraincarcate:una pentru functii membre fara parametri si una

pentru functii membre cu un parametru(mem\_fun\_ref este sablon de functie)

mem\_fun\_ref\_t() cu acelasi rol,avand un t care sugereaza faptul ca este tip

(mem\_fun\_ref\_t este sablon de clasa)

Ex:struct Om{

int varsta;

char sex;

void Scrie() const

{cout<<varsta<<' '<<sex<<'\n';}

void ScrieVarsta(char s) const

{if (sex==s) cout<<varsta<<' ';}

};

int main()

{

Om om[]={{18,'F'},{21,'M'},{16,'F'}};

//Apeleaza Om::Scrie() pentru fiecare obiect Om

for\_each(om,om+3,mem\_fun\_ref(&Om::Scrie));

//Apeleaza Om::ScrieVarsta() pentru fiecare fata

for\_each(om,om+3,bind2nd(mem\_fun\_ref(&Om::ScrieVarsta),'F'));

return 0;

}

Rolul bind2nd in al doilea apel for\_each: leaga 'F' de al doilea argument al mem\_fun\_ref.

Astfel,pentru fiecare element x din sirul om[],se realizeaza apelul x.Om::ScrieVarsta('F').

**Secvente care contin pointeri la obiecte**

mem\_fun (sablon de functie);

mem\_fun\_t (sablon de clasa); //Are acelasi rol

Grupul de sabloane mem\_fun incapsuleaza un pointer la o functie membra.

mem\_fun adapteaza un pointer la o functie membra si returneaza un functor care "stie" sa apeleze

functia membra,ca si cum ar lucra cu o functie nemembra.

**#include <algorithm>**

NOTATII:

-Iteratori: InIt si OutIt -pentru imput iterators si output iterators,

**Algoritmul copy**

int a[]={1, 2, 3, 4 ,5};

vector<int> v(10); ///v=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

///Copiaza elementele 1, 2, 3, 4, 5 in v[0], v[1],...v[4]

copy(a,a+5,v.begin());

FwdIt -pentru forward iterators

BidtIt -pentru bidirectional iterators

RanIt -pentru random access iterators

-Predicate: BinPred -pentru predicate binare

Pred -pentru predicate unare

-Functii si functori:Gen -generatori(f())

UnFunc -functii unare sau functori unari(f(x))

BinFunc-functii binare sau functori binari(f(x,y))

-Operatii: UnOp-operatii unare(functii sau functori)

BinOp-operatii binare(functii sau functori)

**Algoritmi care muta elementele secventei(mutating algorithms)**

Algoritmi Complexitate

copy(fist,last,result) Exact last-first atribuiri

copy\_backward(first,last,result) Exact last-first atribuiri

swap(a,b) Constanta

swap\_ranges(first1,last1,first2) Executa last1-first1 apeluri ale functiei swap()

iter\_swap(it1,it2)

transform(first,last,result,UnOp uop) Executa last-first apeluri ale obiectului

functie uop

transform(first1,last1,first2,result,BinOp bop) Executa last1-first1 apeluri ale obiectului

functie bop

replace(first,last,old\_val,new\_val) Executa last-first apeluri ale predicatului

replace\_if(first,last,Pred pr,new\_val) corespunzator

replace\_copy(first,last,result,old\_val,new\_val) Aplica predicatul corespunzator,de

replace\_copy\_if(first,last,result,Pred pr,new\_val) last-first ori

fill(first,last,val) last-first sau n atribuiri

fill\_n(first,n,val)

generate(first,last,Gen g) last-first sau n apeluri ale generatorului g()

generate\_n(first,n,Gen g) si tot atatea atribuiri

remove(first,last,val) Executa last-first apeluri ale predicatului corespunzator

remove\_if(first,last,Pred pr)

remove\_copy(first,last,result,val) Executa last-first apeluri ale predicatului

remove\_copy\_if(first,last,result,Pred pr) corespunzator

unique(first,last) Executa last-first-1 apeluri ale predicatului

unique(first,last,BinPred pred) corespunzator

unique\_copy(first,last,result) Aplica predicatul corespunzator,de last-first-1 ori

unique\_copy(first,last,result,BinPred pr)

reverse(first,last) Executa (last-first)/2 interschimbari

reverse\_copy(first,last,result) Executa last-first atribuiri

rotate(first,middle,last) Cel mult last-first interschimbari

rotate\_copy(first,middle,last,result) Cel mult last-first atribuiri

random\_shuffle(first,last) Exact last-first-1 interschimbari

random\_shuffle(first,last,RanNumGen& rand)

partition(first,last,Pred pr) Exact last-first aplicari ale predicatului si cel

mult (last-first)/2 interschimbari

stable\_partition(first,last,Pred pr) Exact last-first aplicari ale predicatului si cel

mult (last-first)\*log(last-first) interschimbari

**Idiomul erase-remove**

Daca doriti sa stergeti din container toate elementele cu o anumita valoare,atunci folositi

idiom-ul erase-remove

**Algoritmi care nu modifica secventa(non-modifiying sequence operations)**

Algorimi Complexitate

for\_each(it1,it2,UnFunc) Liniara in n,unde n este it2-it1

count(it1,it2,val) Evalueaza expresia \*it==val de it2-it1 ori

count\_if(it1,it2,Pred) Evalueaza predicatul pr de it2-it1 ori

find(it1,it2,val) Evalueaza expresia \*it==val de cel mult it2-it1 ori

find\_if(it1,it2,Pred) Evalueaza predicatul pr de cel mult it2-it1 ori

search(first1,las1,first2,last2) Aplica == sau predicatul binar,de cel mult

search(first1,last1,first2,last2,BinPred) (last1-first1)\*(last2-first2) ori

search\_n(first,last,n,val) Aplica == sau predicatul binar,de cel mult

search\_n(first,last,n,val,Binpred) (last-first)\*n ori

equal(first1,last1,first2) Aplica == sau predicatul binar,de cel mult

equal(first1,last1,first2,BinPred) (las1-first1) ori

find\_first\_of(first1,last1,first2,last2); Aplica == sau predicatul binar de cel mult

find\_first\_of(first1,last1,first2,last2,BinPred) (last1-first1)\*(last2-first2) ori.

**Algoritmi de sortare sau care efectueaza operatii asociate**

**Agoritmi de sortare**

Algoritmi Complexitate

sort(first,last) In medie,N\*logN comparatii,

sort(first,last,BinPred pr) unde N==last-first

stable\_sort(first,last) Efectueaza cel mult N\*((logN)^2) comparatii,unde

stable\_sort(first,last,BinPred pr) N==last-first.Daca exista suficienta memorie,

atunci,prin utilizarea interna a algoritmului

merge sort,complexitatea se reduce la N\*(logN)

partial\_sort(first,middle,last)

partial\_sort(first,middle,last,BinPred pr) Aproximativ (last-first)\*log(middle-first)

comparatii

partial\_sort\_copy(first,last,result\_first,result\_last) Efectueaza aproximativ

partial\_sort\_copy(first,last,result\_first,result\_last,BinPred pr) (last-first)\*log(min(n1,n2))

comparatii

nth\_element(first,n,last)

nth\_element(first,n,last,BinPred pr) Liniara in medie

**Algoritmi care utilizeaza cautarea binara**

lower\_bound(first,last,val) cel mult log(last-first)+1 comparatii

lower\_bound(first,last,val,BinPred pr)

upper\_bound(first,last,val) cel mult log(last-first)+1 comparatii

upper\_bound(first,last,val,BinPred pr)

equal\_range(first,last,val) cel mult 2\*log(last-first)+1 comparatii

equal\_range(first,last,val,pr)

binary\_search(first,last,val) cel mult log(last-first)+2 comparatii

binary\_search(first,last,val,BinPred pr)

merge(first1,last1,first2,last2,result) cel mult (last1-first1)+(last2-first2)-1

merge(first1,last1,first2,last2,result,BinPred pr) comparatii

inplace\_merge(first,middle,last) In cazul in care exista suficienta memorie

inplace\_merge(first,middle,last,BinPred comp) aditionala,se fac cel mult (last-first)-1

comparatii.Daca nu este memorie aditionala

disponibila,atunci se efectueaza cel mult

N\*logN operatii(N=last-first)

**Operatii pentru structuri sortate,specifice multimilor**

includes(first1,last1,first2,last2) Efectueaza cel mult

includes(first1,last1,first2,last2,BinPred pr) 2\*((last1-first1)+(last2-first2))-1 comparatii

set\_union(first1,last1,first2,last2,result) cel mult 2\*((last1-first1)+(last2-first2))-1

set\_union(first1,last1,first2,last2,result,BinPred pr) comparatii

set\_intersection(first1,last1,first2,last2,result) cel mult 2\*((last1-first1)+(last2-first2))-1

set\_intersection(first1,last1,first2,last2,result,BinPred pr) comparatii

set\_difference(first1,last1,first2,last2,result) cel mult 2\*((last1-first1)+(last2-first2))-1

set\_difference(first1,last1,first2,last2,result,BinPred pr) comparatii

set\_symmetric\_difference(first1,last1,first2,last2,result) cel mult 2\*((last1-first1)+(last2-first2))-1

set\_symmetric\_difference(first1,last1,first2,last2,result,BinPred pr) comparatii

**Operatii cu heap-uri**

make\_heap(first,last)

make\_heap(first,last,BinPred pr) cel mult 3\*logN comparatii, unde N==last-first

push\_heap(first,last)

push\_heap(first,last,BinPred pr) cel mult logN comparatii, unde N==last-first

pop\_heap(first,last)

pop\_heap(first,last,BinPred pr) cel mult 2\*logN comparatii, unde N==last-first

sort\_heap(first,last)

sort\_heap(first,last,BinPred pr) cel mult N\*logN comparatii, unde N==last-first

**Minim si Maxim**

min(a,b)

min(a,b,BinPred pr)

max(a,b)

max(a,b,BinPred pr)

min\_element(first,last)

min\_element(first,last,BinPred pr)

max\_element(first,last) max(last-first-1,0) aplicari ale predicatului

max\_element(first,last,BinPred pr) corespunzator

lexicographical\_compare(first1,last1,first2,last2) cel mult 2\*min(last1-first1,last2-first2)

lexicographical\_compare(first1,last1,first2,last2,BinPred pr) comparatii,folosind predicatul corespunzator

**Generarea permutarilor**

next\_permutation(first,last) cel mult (last-first)/2 interschimbari

next\_permutation(first,last,BinPred pr)

prev\_permutation(first,last) cel mult (last-first)/2 interschimbari

prev\_permutation(first,last,BinPred pr)

**Algoritmi pentru procesare numerica**

**#include <numeric>**

Algoritmi Complexitate

accumulate(first,last,T init) calculeaza suma dintre init si toate

elementele intervalului[first,last),prin

repetarea operatiei init=init+\*it pentru

fiecare iterator it din acest interval

accumulate(first,last,T init,BinOp op) efectueaza in mod repetat

operatia init=op(init,\*it),pentru fiecare

iterator it din intervalul [first,last)

**Produsul scalar**

inner\_product(first1,last1,first2,T init) calculeaza si returneaza produsul

inner\_product(first1,last1,first2,T init,BinOp1 op1,BinOp2 op2) scalar al elementelor a doua

intervale.Se realizeaza in mod

repetat urmatoarele operatii:

init=init+\*i1+\*i2 sau

init=op1(init,op2(\*i1,\*i2)),pentru

fiecare iterator i1 din intervalul

[first,last) si fiecare iterator

i2 din intervalul

[first2,first2+last-first)

Insumarea produselor elementelor

corespunzatoare a doua secvente

se face folosind operator+

si operator\* sau doua operatii

binare op1 si op2,specificate

**Sume partiale**

partial\_sum(first,last,result) efectueaza last+first-1 apeluri ale

partial\_sum(first,last,result,BinOp op) operator+ sau ale operatorului binar op

**Diferenta elementelor adiacente**

adjacent\_difference(first,last,result) efectueaza last+first-1 apeluri ale

adjacent\_difference(first,last,result,BinOp op) operator- sau ale operatorului binar op