



Programare orientata pe obiecte

- suport de curs -

Dobrovat Anca - Madalina

An universitar 2019 – 2020

Semestrul I

Seria 25

Curs 11

11/12/2019



Agenda cursului

1. Biblioteca Standard Template Library - STL
 - Containere, iteratori și algoritmi.
 - Clasele vector, list, map / multimap.



Standard Template Library (STL)

- bibliotecă de clase C++, parte din Standard Library

Ofera:

- structuri de date și algoritmi fundamentali, folosiți la dezvoltarea de programe în C++;
- componente generice, parametrizabile. Aproape toate clasele din STL sunt parametrizate (Template).

Componentele STL se pot compune cu ușurință fără a sacrifica performanța (generic programming)

STL conține clase pentru:

- **containere**
- **iteratori**
- **algoritmi**
- functori (function objects)
- allocators



Standard Template Library (STL)

Container

“Containers are the STL objects that actually store data” (H. Schildt)

- grupare de date în care se pot adauga (insera) si din care se pot sterge (extrage) obiecte;
- gestionează memoria necesară stocarii elementelor, oferă metode de acces la elemente (direct si prin iteratori);
- funcționalități (metode):
 - accesare elemente (ex.: [])
 - gestiune capacitate (ex.: size())
 - modificare elemente (ex.: insert, clear)
 - iterator (begin(), end())
 - alte operații (ie: find)



Standard Template Library (STL)

Container

Tipuri de containere:

- ***de tip secventa*** (in terminologia STL, o secventa este o lista liniara):
 - vector
 - deque
 - list
- ***asociativi*** (permit regasirea eficienta a informatiilor bazandu-se pe chei)
 - set
 - multiset
 - map (permite accesul la informatii cu cheie unica)
 - multimap
- ***adaptor de containere***
 - stack
 - queue
 - priority_queue



Standard Template Library (STL)

Adaugarea si stergerea elementelor din containere:

De tip secventa (vector, deque, list) si - **asociativi** (set, multiset, map, multimap):

- insert()
- erase()

De tip secventa (vector, deque, list) permit si:

- push_back()
- pop_back()

List , Deque

- pop_front()
- push_front()

Containerele asociative:

- find()

Accesarea uzuala: prin iteratori.



Standard Template Library (STL)

Containere de tip secventa

Vector (Dynamic Array)

Vector, Deque, List sunt containere de tip secvență, folosesc reprezentări interne diferite, astfel operațiile uzuale au complexități diferite.

template <class T, class Allocator = allocator <T> > class vector

T = tipul de data utilizat

Allocator = tipul de alocator utilizat (in general cel standard).

- elementele sunt stocate secvențial in zone continue de memorie.

Vector are performanțe bune la:

- Acesare elemente individuale de pe o poziție dată(constant time).
- Iterare elemente în orice ordine (linear time).
- Adăugare/Ștergere elemente de la sfârșit(constant amortized time).



Standard Template Library (STL)

Containere de tip secventa

Vector (Dynamic Array)

Constructori:

```
explicit vector(const Allocator &a = Allocator( ) );
```

*// expl. **vector<int> iv** - vector de int cu zero elemente;*

```
explicit vector(size_type num, const T &val = T ( ), const Allocator &a = Allocator( ));
```

*// expl. **vector<int> iv(10)** - vector de int cu 10 elemente;*

*// expl. **vector<int> iv(10,7)** - vector de int cu 10 elemente, fiecare egal cu 7;*

```
vector(const vector<T,Allocator> &ob);
```

*// expl. **vector<int> iv2(iv)** - vector de int reprezentand copia lui iv;*

```
template <class InIter>
```

```
vector(InIter start, InIter end, const Allocator &a = Allocator( ));
```




Standard Template Library (STL)

Containere de tip secventa

Vector (Dynamic Array)

Functii membre uzuale (sursa H. Schildt)

Member

Description

reference back();

const_reference back() const; returneaza o referinta catre ultimul element;

reference front();

const_reference front() const; returneaza o referinta catre primul element;

iterator begin();

const_iterator begin() const; returneaza un iterator catre primul element;

iterator end();

const_iterator end() const; returneaza un iterator catre ultimul element;



Standard Template Library (STL)

Containere de tip secventa

Vector (Dynamic Array)

Functii membre uzuale (sursa H. Schildt)

Member	Description
<code>void clear();</code>	elimina toate elementele din vector.
<code>bool empty() const;</code>	“true(false)” daca vectorul e (sau nu) gol.
<code>iterator erase(iterator i);</code>	stergerea elementului pointat de i; returneaza un iterator catre elementul de dupa cel sters.
<code>iterator erase(iterator start, iterator end);</code>	stergerea elementelor intre start si end.



Standard Template Library (STL)

Containere de tip secventa

Vector (Dynamic Array)

Functii membre uzuale (sursa H. Schildt)

Member

Description

iterator insert(iterator i, const T &val); - insereaza val inaintea elementului pointati;

void insert(iterator i, size_type num, const T & val) - insereaza un numar de “num” elemente de valoare “val” inaintea elementului pointat de i.

template <class InIter>

void insert(iterator i, InIter start, InIter end); - insereaza o secventa start-end inaintea elementului pointat de i.



Standard Template Library (STL)

Containere de tip secventa

Vector (Dynamic Array)

Functii membre uzuale (sursa H. Schildt)

Member

Description

reference operator[](size_type i) const;
const_reference operator[](size_type i) const; - returneaza o referinta la elementul specificat de i;

void pop_back(); sterge ultimul element.

void push_back(const T &val); adauga la final valoarea "val"

size_type size() const; dimensiunea vectorului.



Standard Template Library (STL)

Containere de tip secventa

Vector (Dynamic Array)

Exemplu citirea unui vector cu accesarea elementelor prin iterator si prin operatorul []

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> v(3);
    v[0] = 12;
    v[1] = 34;
    v[2] = 56;
    //v.resize(5); // necesar pentru reactualizarea dimensiunii vectorului
    v[3] = 78;
    v[4] = 90;
    for(int i = 0; i<v.size(); i++)
        cout<<v[i]<<" ";
}
```



Standard Template Library (STL)

Containere de tip secventa

Vector (Dynamic Array)

Exemplu citirea unui vector cu accesarea elementelor prin iterator si prin operatorul []

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> v(3);
    v.push_back(12);
    v.push_back(34);
    v.push_back(56);
    v.push_back(78);
    v.push_back(90);
    for(int i = 0; i<v.size(); i++)
        cout<<v[i]<<" ";
    // Afisare 8 valori, resize automat
}
```



Standard Template Library (STL)

Containere de tip secventa

Vector (Dynamic Array)

Exemplu

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<char> v(10);
    unsigned int i;
    cout << "Size = " << v.size() << endl;
    for(i=0; i<10; i++) v[i] = i + 'a';
    for(i=0; i<v.size(); i++) cout << v[i] << " ";
    for(i=0; i<10; i++) v.push_back(i + 10 + 'a');
    cout << "Size now = " << v.size() << endl;
    for(i=0; i<v.size(); i++) cout << v[i] << " ";
    for(i=0; i<v.size(); i++) v[i] = toupper(v[i]);
    for(i=0; i<v.size(); i++) cout << v[i] << " ";
}
```



Standard Template Library (STL)

Containere de tip secventa

Vector (Dynamic Array)

Exemplu - accesarea unui vector cu iterator

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<char> v(10);
    vector<char>::iterator p;

    for( p = v.begin(); p != v.end(); p++)
        *p = i + 'a';

    for( p = v.begin(); p != v.end(); p++)        cout << *p << " ";

    for( p = v.begin(); p != v.end(); p++)
        *p = toupper(*p);
}
```




Standard Template Library (STL)

Containere de tip secventa

Vector (Dynamic Array)

Exemplu - inserarea si stergerea elementelor intr-un vector

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<char> v(10);
    vector<char> v2;
    char str[] = "<Vector>";
    unsigned int i;

    for(i=0; i<10; i++) v[i] = i + 'a';
    for(i=0; str[i]; i++) v2.push_back(str[i]);
    for(i=0; i<v.size(); i++) cout << v[i] << " ";
}
```



Standard Template Library (STL)

Containere de tip secventa

Vector (Dynamic Array)

Exemplu - inserarea elementelor intr-un vector

```
#include<vector>
```

```
vector<int> v(3);
```

```
v[0] = 12;
```

```
v[1] = 34;
```

```
v[2] = 56; /// 12,34,56
```

```
v.resize(5);
```

```
v[3] = 78;
```

```
v[4] = 90; /// 78, 90
```

```
vector<int>::iterator p;
```

```
p = v.begin();
```

```
p++;
```

```
v.insert(p,5,100); /// 12, 100, 100, 100, 100, 100, 34, 56, 78, 90
```



Standard Template Library (STL)

Containere de tip secventa

Vector (Dynamic Array)

Exemplu - inserarea elementelor intr-un vector

```
#include<vector>
```

```
vector<int> v(3);
```

```
v[0] = 12;
```

```
v[1] = 34;
```

```
v[2] = 56; /// 12,34,56
```

```
v.resize(5);
```

```
v[3] = 78;
```

```
v[4] = 90; /// 78, 90
```

```
vector<int>::iterator p;
```

```
p = v.begin();
```

```
p++;
```

```
v.erase(p); /// 34, 56, 78, 90
```

```
v.erase(p,p+2); /// 78, 90
```



Standard Template Library (STL)

Containere de tip secventa

Vector (Dynamic Array)

Exemplu - inserarea elementelor de tip definit de utilizator

```
class Test { int i;  
public:  
Test(int x = 0):i(x){cout<<"C ";}  
Test(const Test& x){i = x.i; cout<<"CC ";}  
~Test(){cout<<"D ";} }  
  
int main() {  
vector<Test> v;  
v.push_back(10); cout<<endl;   /// C CC D  
v.push_back(20); cout<<endl;   /// C CC CC D D  
Test ob(30);  
v.push_back(ob); cout<<endl;   /// C CC CC CC D D  
Test& ob2 = ob;  
v.push_back(ob2);   /// CC D D D D  
}
```



Standard Template Library (STL)

Containere de tip secventa

List

template <class T, class Allocator = allocator <T> > class list

T = tipul de data utilizat

Allocator = tipul de alocator utilizat (in general cel standard).

- implementat ca și listă dublă înlănțuită

List are performanțe bune la:

- Ștergere/adăugare de elemente pe orice poziție (constant time).
- Mutarea de elemente sau secvențe de elemente în liste sau chiar și între liste diferite(constant time).
- Iterare de elemente in ordine (linear time)



Standard Template Library (STL)

Containere de tip secventa

List

Constructori:

```
explicit list(const Allocator &a = Allocator( ) );  
// expl. list<int> iv
```

```
explicit list(size_type num, const T &val = T( ), const Allocator &a =  
Allocator());  
// expl. list<int> iv(10);  
// expl. list<int> iv(10,7);
```

```
list(const list <T,Allocator> &ob);  
// expl. list<int> iv2(iv);
```

```
template <class InIter>  
list(InIter start, InIter end, const Allocator &a = Allocator( ));
```



Standard Template Library (STL)

Containere de tip secventa

List

Functii membre uzuale (sursa H. Schildt)

Member	Description
reference back(); const_reference back() const;	returneaza o referinta catre ultimul element;
reference front(); const_reference front() const;	returneaza o referinta catre primul element;
iterator begin(); const_iterator begin() const;	returneaza un iterator catre primul element;
iterator end(); const_iterator end() const;	returneaza un iterator catre ultimul element;



Standard Template Library (STL)

Containere de tip secventa

List

Functii membre uzuale (sursa H. Schildt)

Member	Description
<code>void clear();</code>	elimina toate elementele din vector.
<code>bool empty() const;</code>	“true(false)” daca vectorul e (sau nu) gol.
<code>iterator erase(iterator i);</code>	stergerea elementului pointat de i; returneaza un iterator catre elementul de dupa cel sters.
<code>iterator erase(iterator start, iterator end);</code>	stergerea elementelor intre start si end.



Standard Template Library (STL)

Containere de tip secventa

List

Functii membre uzuale (sursa H. Schildt)

Member

Description

iterator insert(iterator i, const T &val); - insereaza val inaintea elementului pointat;

void insert(iterator i, size_type num, const T & val) - insereaza un numar de “num” elemente de valoare “val” inaintea elementului pointat de i.

template <class InIter>

void insert(iterator i, InIter start, InIter end); - insereaza o secventa start-end inaintea elementului pointat de i.



Standard Template Library (STL)

Containere de tip secventa

List

Functii membre uzuale (sursa H. Schildt)

Member

Description

`void merge(list <T, Allocator> &ob);` - concateneaza lista din ob; aceasta din urma devine vida.

`template <class Comp>`

`void merge(list &ob, Comp cmpfn);` - concateneaza si sorteaza;

`void pop_back();`

sterge ultimul element.

`void pop_front();`

sterge primul element.

`void push_back(const T &val);`

adauga la final valoarea "val"

`void push_front(const T &val);`

adauga la inceput valoarea "val"



Standard Template Library (STL)

Containere de tip secventa

List

Functii membre uzuale (sursa H. Schildt)

Member	Description
<code>void remove(const T &val);</code>	- elimina toate valorile “val” din lista;
<code>void reverse();</code>	- inverseaza lista.
<code>size_type size() const;</code>	- numarul de elemente.
<code>void sort();</code> <code>template <class Comp></code> <code>void sort(Comp cmpfn);</code>	- sortarea cu o functie de comparatie.



Standard Template Library (STL)

Containere de tip secventa

List

Exemplu

```
#include <iostream>
#include <list>
using namespace std;
int main()
{
    list<int> lst; // create an empty list
    int i;
    for(i=0; i<10; i++) lst.push_back(i);
    cout << "Size = " << lst.size() << endl;
    list<int>::iterator p;
    for(p = lst.begin(); p != lst.end(); p++)        cout << *p << " ";
    for(p = lst.begin(); p != lst.end(); p++)
        *p = *p + 100;
    for(p = lst.begin(); p != lst.end(); p++)        cout << *p << " ";
    return 0;
}
```



Standard Template Library (STL)

Containere de tip secventa

List

Exemplu

```
list<int> lst; // lista vida
for(int i=0; i<10; i++) lst.push_back(i); // insereaza 0 .. 9

// Lista afisata in ordine inversa - mod 1"
list<int>::iterator p;
p = lst.end();
while(p != lst.begin()) {
    p--; // decrement pointer before using
    cout << *p << " ";
}

// Lista afisata in ordine inversa - mod 2"
list<int>::reverse_iterator q;
for(q = lst.rbegin(); q!=lst.rend();q++)
    cout<<*q<<" ";
```



Standard Template Library (STL)

Containere de tip secventa

List

Exemplu - push_front, push_back si sort

```
#include <iostream>
#include <list>
using namespace std;
int main() {
    list<int> lst1, lst2;
    int i;
    for(i=0; i<10; i++) lst1.push_back(i);
    for(i=0; i<10; i++) lst2.push_front(i);

    list::iterator p;
    for( p = lst1.begin(); p != lst1.end(); p++) cout << *p << " ";
    for( p = lst2.begin(); p != lst2.end(); p++) cout << *p << " ";

    // sort the list
    lst1.sort();
}
```



Standard Template Library (STL)

Containere de tip secventa

List

Exemplu - ordonare crescatoare si descrescatoare

```
#include<iostream>
```

```
#include<list>
```

```
using namespace std;
```

```
bool compare(int a, int b) {return a>b;}
```

```
int main() {
```

```
list<int> l;
```

```
l.push_back(5); l.push_back(3); l.push_front(2);
```

```
list<int>::iterator p;
```

```
l.sort(); //crescator
```

```
for(p=l.begin(); p!=l.end();p++) cout<<*p<<" ";
```

```
l.sort(compare); // descrescator prin functia comparator
```

```
for(p=l.begin(); p!=l.end();p++) cout<<*p<<" ";
```

```
}
```



Standard Template Library (STL)

Containere de tip secventa

List

Exemplu - concatenarea a 2 liste

```
#include <iostream>
#include <list>
using namespace std;
int main()
{
    list<int> lst1, lst2; /*...create liste ...*/
    list<int>::iterator p;
    // concatenare
    lst1.merge(lst2);
    if(lst2.empty())
        cout << "lst2 vida\n";
    cout << "Contents of lst1 after merge:\n";
    for(p = lst1.begin(); p != lst1.end(); p++) cout << *p << " ";
    return 0;
}
```




Standard Template Library (STL)

Containere de tip secventa

List

Exemplu - stocarea obiectelor in list

```
class Test { int i;  
public:  
Test(int x = 0):i(x){cout<<"C "};  
Test(const Test& x){i = x.i; cout<<"CC "};  
~Test(){cout<<"D ";} }  
  
int main() {  
list<Test> v;  
v.push_back(10); cout<<endl;  /// C CC D  
v.push_back(20); cout<<endl;  /// C CC D  
Test ob(30);  
v.push_back(ob); cout<<endl;  /// C CC  
Test& ob2 = ob;  
v.push_back(ob2);  /// CC D D D D D  
}
```



Standard Template Library (STL)

Containere de tip secventa

List

Exemplu - stocarea obiectelor in list

```
#include <iostream>
#include <list>
#include <cstring>
using namespace std;
class myclass
{
    int a, b, sum;
public:
    myclass(){a = b = 0;}
    myclass(int i, int j){a = i;b = j;sum = a + b;}
    int getsum(){return sum;}
    friend bool operator<(myclass &o1,myclass &o2) {return o1.sum < o2.sum;}
    friend bool operator>(myclass &o1,myclass &o2) {return o1.sum > o2.sum;}
    friend bool operator==(myclass &o1,myclass &o2){return o1.sum == o2.sum;}
    friend bool operator!=(myclass &o1,myclass &o2){return o1.sum != o2.sum;}
};
```



Standard Template Library (STL)

Containere de tip secventa

List

Exemplu - stocarea obiectelor in list

```
int main() {  
    int i;  
    list<myclass> lst1;  
    list<myclass>::iterator p;  
  
    for(i=0; i<10; i++) lst1.push_back(myclass(i, i));  
    for (p = lst1.begin(); p != lst1.end(); p++) cout << p->getsum() << " ";  
    cout << endl;  
  
    // create a second list  
    list<myclass> lst2;  
    for(i=0; i<10; i++) lst2.push_back(myclass(i*2, i*3));  
    for (p = lst2.begin(); p != lst2.end(); p++) cout << p->getsum() << " ";  
    cout << endl;  
  
    // now, merget lst1 and lst2  
    lst1.merge(lst2);  
    return 0;  
}
```



Standard Template Library (STL)

Containere de tip secventa

Deque (double ended queue) - Coadă dublă (completă)

- elementele sunt stocate în blocuri de memorie (chunks of storage)
- elementele se pot adăuga/șterge eficient de la ambele capete

Vector vs Deque

- Accesul la elemente de pe orice poziție este mai eficient la vector
- Inserare/ștergerea elementelor de pe orice poziție este mai eficient la Deque (dar nu e timp constant)
- Pentru liste mari Vector alocă zone mari de memorie, deque alocă multe zone mai mici de memorie – Deque este mai eficient în gestiunea memoriei



Standard Template Library (STL)

Adaptor de container

- Încapsulează un container de tip secvență, și folosesc acest obiect pentru a oferi funcționalități specifice containerului (stivă, coadă, coadă cu priorități).

Stack: strategia LIFO (last in first out) pentru adaugare/ștergere elemente

Operații: empty(), push(), pop(), top().

```
template < class T, class Container = deque<T> > class stack;
```

Queue: strategia FIFO (first in first out)

Operații: empty(), front(), back(), push(), pop(), size();

```
template < class T, class Container = deque<T> > class queue;
```

Priority_queue: se extrag elemente pe baza priorităților

Operații: empty(), top(), push(), pop(), size();

```
template < class T, class Container = vector<T>, class Compare =  
less<typename Container::value_type> > class priority_queue;
```



Standard Template Library (STL)

Adaptor de container

Exemplu stiva

```
#include <stack>
#include<iostream>
using namespace std;
void sampleStack() {
    stack<int> s;
    s.push(3);    s.push(4);    s.push(1);    s.push(2);
    while (!s.empty()) {
        cout << s.top() << " ";
        s.pop();    }
}
int main() {
    sampleStack();
}
```



Standard Template Library (STL)

Adaptor de container

Exemplu stiva

```
#include <stack>
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {
```

```
    stack<int,vector<int>> s; // primul parametru = tipul elementelor, al doilea  
    parametru, stilul de stocare
```

```
    s.push(1);
```

```
    s.push(3);
```

```
    s.push(5);
```

```
    for(; !s.empty();)
```

```
    {
```

```
        cout<<s.top()<<endl;
```

```
        s.pop();
```

```
    }
```

```
}
```



Standard Template Library (STL)

Adaptor de container

Exemplu stiva - influenta celui de-al doilea parametru (eventual)

```
class Test { int i;  
public:  
Test(int x = 0):i(x){cout<<"C ";}  
Test(const Test& x){i = x.i; cout<<"CC ";}  
~Test(){cout<<"D ";}  
  
int main()  
{  
    stack<Test> s;  
    s.push(1); // C CC D  
    cout<<endl;  
    s.push(3); // C CC D  
    cout<<endl;  
    s.push(5); // C CC D urmat de D D D (distrugerea listei)  
}
```




Standard Template Library (STL)

Adaptor de container

Exemplu stiva - influenta celui de-al doilea parametru (eventual)

```
class Test { int i;  
public:  
Test(int x = 0):i(x){cout<<"C ";}  
Test(const Test& x){i = x.i; cout<<"CC ";}  
~Test(){cout<<"D ";}};  
  
int main()  
{  
    stack<Test, vector<Test>> s;  
    s.push(1); // C CC D  
    cout<<endl;  
    s.push(3); // C CC CC D D  
    cout<<endl;  
    s.push(5); // C CC CC CC D D D urmat de D D D (distrugerea listei)  
}
```



Standard Template Library (STL)

Adaptor de container

Exemplu stiva - influenta celui de-al doilea parametru (eventual)

```
class Test { int i;  
public:  
Test(int x = 0):i(x){cout<<"C "};  
Test(const Test& x){i = x.i; cout<<"CC "};  
~Test(){cout<<"D "};  
  
int main()  
{  
    stack<Test, list<Test>> s;  
    s.push(1); // C CC D  
    cout<<endl;  
    s.push(3); // C CC D  
    cout<<endl;  
    s.push(5); // C CC D urmat de D D D (distrugerea listei)  
}
```



Standard Template Library (STL)

Adaptor de container

Exemplu coada

```
#include <queue>
#include<iostream>
using namespace std;
void sampleQueue() {
    queue<int> s;
    //queue<int,deque<int>>
    //queue<int,list<int> >
    s.push(3);    s.push(4);    s.push(1);    s.push(2);
    while (!s.empty()) {
        cout << s.front() << " ";
        s.pop();    }
}
int main() {
    sampleQueue();
}
```



Standard Template Library (STL)

Adaptor de container

Exemplu coada cu prioritate

```
#include <queue>
#include<iostream>
using namespace std;
void samplePriorQueue() {
    priority_queue<int> s;
    //priority_queue<int,deque<int>>
    //priority_queue<int,list<int> >
    s.push(3);    s.push(4);    s.push(1);    s.push(2);
    while (!s.empty()) {
        cout << s.top() << " ";
        s.pop();    }
}
int main() {
    samplePriorQueue();
}
```



Standard Template Library (STL)

Containere asociative

eficiente în accesare elementelor folosind chei (nu folosind poziții ca și în cazul containerelor de tip secvență).

- **Set**

- mulțime - stochează elemente distincte.
- se folosește arbore binar de căutare ca și reprezentare internă

- **Map**

- dictionar - stochează elemente formate din cheie și valoare
- nu putem avea chei duplicate
- **multimap** poate avea chei duplicate

- **Bitset**

- container special pentru a stoca biti.



Standard Template Library (STL)

Containere asociative

Map

- in sens general, map = lista de perechi cheie - valoare

template <class Key, class T, class Comp = less<Key>, class Allocator = allocator <pair<const Key,T> > class map

Key = tipul cheilor

T = tipul valorilor

Comp = functie care compara 2 chei

Allocator = tipul de alocator utilizat (in general cel standard).

Inserarea se face ordonat dupa chei.



Standard Template Library (STL)

Containere asociative

Map

Constructori:

```
explicit map(const Comp &cmpfn = Comp( ), const Allocator &a = Allocator( ) );
```

```
// expl. map<char,int> m; - map cu zero elemente;
```

```
map(const map<Key,T,Comp,Allocator> &ob);
```

```
template <class InIter>
```

```
map(InIter start, InIter end, const Comp &cmpfn = Comp( ), const Allocator &a  
= Allocator( ));
```



Standard Template Library (STL)

Containere asociative

Map

Functii membre uzuale (sursa H. Schildt)

Member	Description
<code>iterator begin();</code> <code>const_iterator begin() const;</code>	returneaza un iterator catre primul element;
<code>iterator end();</code> <code>const_iterator end() const;</code>	returneaza un iterator catre ultimul element;
<code>void clear();</code>	elimina toate elementele din map.
<code>size_type count(const key_type &k) const;</code>	- numarul de aparitii ale lui k
<code>bool empty() const;</code>	“true(false)” daca vectorul e (sau nu) gol.



Standard Template Library (STL)

Containere asociative

Map

Functii membre uzuale (sursa H. Schildt)

Member

Description

`iterator erase(iterator i);` stergerea elementului pointat de `i`; returneaza un iterator catre elementul de dupa cel sters.

`iterator erase(iterator start, iterator end);` stergerea elementelor intre `start` si `end`.

`size_type erase(const key_type &k)` - stergerea tuturor valorilor `k`

`iterator find(const key_type &k);`

`const_iterator find(const key_type &k) const;` - returneaza un iterator catre valoarea cautata `k` sau catre sfarsitul map daca nu exista.



Standard Template Library (STL)

Containere asociative

Map

Functii membre uzuale (sursa H. Schildt)

Member

Description

`size_type size() const;`

`iterator insert(iterator i, const value_type &val);` - insereaza val pe pozitia pointata de i sau dupa;

`template <class InIter>`

`void insert(InIter start, InIter end);` - insereaza o secventa start-end.

`pair <iterator,bool>`

`insert(const value_type &val);`

`mapped_type & operator[](const key_type &i)` - returneaza o referinta la elementul cheie i, daca nu exista, atunci se insereaza.



Standard Template Library (STL)

Containere asociative

Map

Exemplu

```
#include <iostream>
#include <map>
using namespace std;
int main() {
    map<int, int> m;
    m.insert(pair<int, int>(1,100));    m.insert(pair<int, int>(7,300));
    m.insert(pair<int, int>(3,200));    m.insert(pair<int, int>(5,400));
    m.insert(pair<int, int>(2,500));

    map<int, int>::iterator p;
    for(p = m.begin(); p!= m.end(); p++)
        cout<<p->first<<" "<<p->second<<endl;

    //copierea intr-un alt map
    map<int,int>m2(m.begin(),m.end());
}
```



Standard Template Library (STL)

Containere asociative

Map

Exemplu

```
#include <iostream>
#include <map>
using namespace std;
int main() {
    map<char, int> m;
    for(int i=0; i<26; i++)
        m.insert(pair<char, int>('A'+i, 65+i));

    char ch;    cout << "Enter key: ";    cin >> ch;
    map<char, int>::iterator p;
    // find value given key
    p = m.find(ch);
    if(p != m.end())    cout << "Its ASCII value is " << p->second;
    else    cout << "Key not in map.\n";
    return 0;
}
```



Standard Template Library (STL)

Containere asociative

Multimap

Exemplu - catalog studenti (cheia = nume si valoarea = numar matricol)

```
#include <iostream>
#include <map>
using namespace std;
int main( ) {
    multimap<string, int> m;
    m.insert(m.end(), make_pair("Ionescu",100));
    // echivalent cu m.insert(pair<string,int>("Ionescu",100));
    m.insert(m.end(), make_pair("Popescu",355));
    m.insert(m.end(), make_pair("Ionescu",234));

    map<string, int>::iterator p;
    for(p = m.begin(); p!= m.end(); p++)
        cout<<p->first<<" "<<p->second<<endl;
```



Standard Template Library (STL)

Containere asociative

Multimap

Exemplu - catalog studenti (cheia = nume si valoarea = numar matricol)

//Afisarea elementelor cu un anumit nume

```
string nume;  
cin>>nume;  
for(p = m.begin(); p!= m.end(); p++)  
if (p->first == nume)  
    cout<<p->first<<" "<<p->second<<endl;
```

//stergerea primei aparitii a unui nume

```
cin>>nume;  
map<string, int>::iterator f;  
f = m.find(nume);  
if (f!=m.end()) m.erase(f);  
for(p = m.begin(); p!= m.end(); p++)  
    cout<<p->first<<" "<<p->second<<endl;
```



Standard Template Library (STL)

Containere asociative

Multimap

Exemplu - catalog studenti (cheia = nume si valoarea = numar matricol)

```
//Stergerea tuturor aparitiilor unui nume  
cin>>nume;  
m.erase(nume);
```



Standard Template Library (STL)

Iterator

- un concept fundamental in STL, este elementul central pentru algoritmi oferiti de STL;
- obiect care gestionează o poziție (curentă) din containerul asociat;
- suport pentru traversare (++/--), dereferențiere (*it);
- permite decuplarea intre algoritmi si containere.

Tipuri de iteratori:

- iterator input/output (istream_iterator, ostream_iterator)
- forward iterators, iterator bidirectional, iterator random access
- reverse iterators.

Adaptoarele de containere (container adaptors) - stack, queue, priority_queue - nu oferă iterator!



Standard Template Library (STL)

Iterator

Functii uzuale:

- `begin()` - returneaza pozitia de inceput a containerului
- `end()` - returneaza pozitia de dupa terminarea containerului
- `advance()` - incrementeaza pozitia iteratorului
- `next()` - returneaza noul iterator dupa avansarea cu o pozitie
- `prev()` - returneaza noul iterator dupa decrementarea cu o pozitie
- `inserter()` - insereaza elemente pe orice pozitie



Standard Template Library (STL)

Iterator

Exemplu - begin, end, advance, next, prev:

```
vector<int> v = { 1, 2, 3, 4, 5 };  
    vector<int>::iterator p;  
    for (p = v.begin(); p < v.end(); p++)  
        cout << *p << " ";  
    cout<<endl;  
    p = v.begin();  
    advance(p, 2);  
    vector<int>::iterator n = next(p);  
    vector<int>::iterator d = prev(p);  
    for (; p < v.end(); p++)  
        cout << *p << " ";  
    cout<<endl<<*n<<" "<<*d;
```



Standard Template Library (STL)

Iterator

Exemplu - inserter:

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> v = { 1, 2, 3, 4, 5 };
    vector<int> v2 = { 10, 20, 30};
    vector<int>::iterator p= v.begin();
    advance(p, 2);
    copy(v2.begin(),v2.end(),inserter(v,p));
    for (p = v.begin(); p < v.end(); p++)
        cout << *p << " ";
    return 0;
}
```



Standard Template Library (STL)

Reverse iterator

Exemplu

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> v = { 1, 2, 3, 4, 5 };

    vector<int>::iterator p;
    for (p = v.begin(); p < v.end(); p++)
        cout << *p << " ";
    cout<<endl;
    vector<int>::reverse_iterator r;
    for (r = v.rbegin(); r < v.rend(); r++)
        cout << *r << " ";
    return 0;
}
```



Standard Template Library (STL)

Algoritm

- colecție de funcții template care pot fi folosite cu iteratori. Funcțiile operează pe un domeniu (range) definit folosind iteratori;
- domeniu (range) este o secvență de obiecte care pot fi accesate folosind iteratori sau pointeri.
- headere: `<algorithm>`, `<numeric>`



Standard Template Library (STL)

Algoritm

- Operații pe secvențe
 - care nu modifică sursa: accumulate, count, find, count_if, etc
 - care modifică : copy, transform, swap, reverse, random_shuffle, etc.
- Sortări: sort, stable_sort, etc.
- Pe secvențe de obiecte ordonate
 - Căutare binară : binary_search, etc
 - Interclasare (Merge): merge, set_union, set_intersect, etc.
- Min/max: min, max, min_element, etc.
- Heap: make_heap, sort_heap, etc.



Standard Template Library (STL)

Algoritm

Exemplu: – accumulate : calculează suma elementelor

```
vector<int> v;  
v.push_back(3);  
v.push_back(4);  
v.push_back(2);  
v.push_back(7);  
v.push_back(17);  
//compute the sum of all elements in the vector  
cout << accumulate(v.begin(), v.end(), 0) << endl;  
//compute the sum of elements from 1 inclusive, 4 exclusive [1,4)  
vector<int>::iterator start = v.begin()+1;  
vector<int>::iterator end = v.begin()+4;  
cout << accumulate(start, end, 0) << endl;
```



Standard Template Library (STL)

Algoritm

Exemplu: – copy

```
vector<int> v;  
v.push_back(3);  
v.push_back(4);  
v.push_back(2);  
v.push_back(7);  
v.push_back(17);  
//make sure there are enough space in the destination  
//allocate space for 5 elements  
vector<int> v2(5);  
//copy all from v to v2  
copy(v.begin(), v.end(), v2.begin());
```




Standard Template Library (STL)

Algoritm

Exemplu: – sort

Sorteaza elementele din intervalul [first,last) ordine crescătoare.

- Elementele se compară folosind operatorul operator <

```
vector<int> v;  
v.push_back(3);  
v.push_back(4);  
v.push_back(2);  
v.push_back(7);  
v.push_back(17);
```

```
sort(v.begin(), v.end());
```



Standard Template Library (STL)

Algoritm

Exemplu: – sort cu o functie de comparare

```
bool asc(int i, int j) {  
    return (i < j);  
}
```

```
vector<int> v;  
v.push_back(3);  
v.push_back(4);  
v.push_back(2);  
v.push_back(7);  
v.push_back(17);
```

```
sort(v.begin(), v.end(), asc);
```



Standard Template Library (STL)

Algoritm

Exemplu: – for_each

```
void print(int elem) {  
    cout << elem << " ";  
}  
void testForEach() {  
    vector<int> v;  
    v.push_back(3);  
    v.push_back(4);  
    v.push_back(2);  
    v.push_back(7);  
    v.push_back(17);  
  
    for_each(v.begin(), v.end(), print);  
    cout << endl;  
}
```



Standard Template Library (STL)

Algoritm

Exemplu: – transform

aplică funcția pentru fiecare element din secvență ([first1,last1))
rezultatul se depune în secvența rezultat

```
int multiply3(int el) {  
    return 3 * el;  
}  
void testTransform() {  
    vector<int> v;  
    v.push_back(3); v.push_back(4); v.push_back(2);v.push_back(7);  
    v.push_back(17);  
  
    vector<int> v2(5);  
    transform(v.begin(), v.end(), v2.begin(), multiply3);  
    //print the elements  
    for_each(v2.begin(), v2.end(), print);  
}
```



Standard Template Library (STL)

Functor - STL Function Objects

Functor – Orice clasă ce are definit operator ()

Față de pointer la funcție, functorul poate avea informații adiționale (variabile membre)

Exemplu:

```
class MyClass {  
public:  
    bool operator()(int i, int j) {  
        return (i > j);  
    }  
};  
sort(v.begin(), v.end(), MyClass());
```



Perspective

Cursul 12:

Șabloane de proiectare (Design Pattern)

1. Definiție și clasificare.
2. Exemple de șabloane de proiectare (Singleton, Abstract Object Factory).