Programare orientată obiect

Cursurile 12 și 13

rev 1.0/PP/20131219

Sumar

- Biblioteca C++ de clase și funcții generice standard (STL – Standard Template Library)
  - Containere
  - Iteratori
  - Algoritmi

STL

- Bibliotecă de clase și funcții **generice** C++
- Implementează:
  - Containere
    - Diferite structuri de date
  - Algoritmi:
    - Colecție de funcții de prelucrare a elementelor containerelor
    - Operează cu iteratori sau pointeri
  - Iteratori:
    - Obiecte de acces la componentele anumitor containere
    - Transmise ca parametri algoritmilor

Containere

- Secvențiale
- Asociative
- Adaptive

Containere secvențiale

- Colecții liniare și ordonate de date
- Accesul la elemente se face pe baza poziției acestora în cadrul containerului
- Clase predefinite:
  - vector
  - deque
  - list
- Elementele se regăsesc în ordinea în care ele sunt adăugate
- Constructori:
  - Impliciți
  - Cu un număr precizat de elemente (aceeași valoare)
  - Cu elementele preluate dintr-un alt container
  - De copiere

Funcții

- Predicate unare
  - bool functie(param);
- Predicate binare
  - bool functie(param1, param2);
- Functori

```
class Functor
{
public:
    bool operator()(param);//sau
    bool operator()(param1, param2);
}
```

Iteratori (1)

- Obiecte prin intermediul cărora sînt referite alte obiecte
- Asemănători pointerilor
- Utilizați pentru accesul la elementele anumitor containere
- Interfață de comunicație între algoritmi și containere
  - Preluați ca parametri de către algoritmi
  - Modatitate de acces la elementele containerelor
- Traversare: înainte, înainte și inapoi
- Operații: citire, scriere, citire și scriere
- Acces: secvențial și direct

Iteratori (1)

De intrare	**De ieșire**
De intrare/ieșire (forward)	
Bidirecționali	
Pentru acces direct	

Iteratori: inițializare

- Definiți de programator
  - container<T>::iterator **it**;
  - container<T>::const_iterator **c_it**;
  - container<T>::reverse_iterator **r_it**;
    - **it** este un iterator pentru containerul exemplificat
    - **c_it** este un iterator constant pentru containerul exemplificat
    - **r_it** este un iterator invers pentru containerul exemplificat
- Metode în container (returnează iteratorul corespunzător)
  - begin(), rbegin()
  - end(), rend()
  - Exemplu:
    - it = **cont.begin()**;
    - it != **cont.end();**

Iteratori - Operații

Operație	Intrare	Ieșire	Intrare/Ieșire	Bidirecționali	Acces direct	Exemplu
++	*	*	*	*	*	it++, ++it
=	*	*	*	*	*	it1 = it2
==, !=	*		*	*	*	it1 == it2
*** (rvalue), ->**	*		*	*	*	*it
***(lvalue)**		*	*	*	*	*it = val
--				*	*	it--, --it
+, -, >, >=, <, <=					*	it + d, d + it
[]					*	it[d]

Clasa vector

- Spațiu contiguu de memorie
- Alocare dinamică
- Eficient
  - Inserări/ștergeri la sfîrșit
  - Adresarea directă a elementelor
- include <vector>

Clasa vector

- Inserare
  - **push_back**(val)
  - **insert**(iterator, valoare)
  - **insert**(iterator, numar_elemente_de_inserat, valoare)
- Ștergere
  - **erase**(iterator)
  - **pop_back**()
- Referirea elementelor (set/get):
  - operatorul **[]**(index)
  - metoda **at**(index)
  - metodele **front**() și **back**()
  - iteratori

Clasa vector

- Dimensiunea actuală
  - size_t **size**()
- Numărul maxim de elemente
  - size_t **capacity**()
- Ștergerea tuturor elementelor:
  - void **clear**()
- Verificare existență elemente
  - bool **empty**()
- Interschimbul elementelor a doi vectori de același tip:
  - void **swap**(vector &)
- Operatori relaționali
  - <, <=, >, >=, ==, !=

Clasa vector

```
vector<int> vInt;

vInt.push_back(10);
vInt.push_back(20);

for( unsigned int i =0; i <vInt.size(); i++)
{
        cout<<vInt[i]<<endl;
}
```

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

for(it = vInt.begin(); it != vInt.end(); it++)
{
        cout<<*it<<endl;
}
vInt.pop_back();

it = vInt.begin();
vInt.insert(it, 5, 0);
```

Clasa deque

- Coadă cu două capete
- Access direct
  - ca spațiu contiguu de memorie
- Alocare dinamică
- Permite adăugarea elementelor la ambele capete
- Parcurgere în ambele direcții
- Timp constant pentru:
  - Inserare
  - Ștergere
- include <deque>

Clasa deque

- Metodele asemănătoare clasei vector
  - Mai puțin **capacity**()
- Metode de acces la început
  - **push_front**()
  - **pop_front**()

Clasa deque

```
deque<int> dInt;

dInt.push_back(10);
dInt.push_back(20);
dInt.push_back(30);
dInt.push_front(-40);
dInt.push_front(-50);
dInt.push_front(-60);
```

```
deque<int>::const_iterator dit;

for(dit = dInt.begin(); dit != dInt.end(); dit++)
{
        cout<<*dit<<endl;
}

dInt.pop_front();
dInt.pop_back();
```

Clasa list

- Spațiu necontiguu de memorie
- Alocare dinamică
- Listă dublu înlănțuită
- Permite adăugarea elementelor la ambele capete
- Elementele nu pot fi accesate direct (**[]**)
- Parcurgere în ambele direcții
- Timp constant pentru:
  - Inserții
  - Ștergeri
- include <list>

Clasa list

- Inserare
  - **push_back**(val)
  - **push_front**(val)
  - **insert**(iterator, valoare)
  - **insert**(iterator, numar_elemente_de_inserat, valoare)
- Ștergere
  - **erase**(iterator)
  - **pop_back**()
  - **pop_front**()
- Referirea elementelor (set/get):
  - metodele **front**() și **back**()
  - iteratori

Clasa list

- Dimensiunea actuală
  - **size**()
- Ștergerea tuturor elementelor:
  - **clear**()
- Ștergerea elementelor egale cu o valoare
  - **remove**()
- Interschimbul elementelor a două liste de același tip:
  - **swap**(list &)

Clasa list

- Sortarea elemenetelor
  - **sort**()
  - **sort**(comparator)
- Schimbarea ordinii elementelor
  - **reverse**()
- Concatenarea listelor sortate cu eliminarea elementelor din sursă
  - **merge**(list&)
- Eliminarea valorilor duplicate
  - **unique**()
  - **unique**(comparator)

Clasa list

```
list<int> lInt;

lInt.push_back(10);
lInt.push_back(20);
lInt.push_back(30);
lInt.push_front(-40);
lInt.push_front(-50);
lInt.push_front(-60);
```

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

for(lit = lInt.begin(); lit != lInt.end(); lit++)
{
        cout<<*lit<<endl;
}
lInt.clear();
```

Containere asociative

- Stocarea elementelor se face pe baza unor chei
- Accesul se realizează direct, după cheie
- Elementele sunt stocate în ordinea cheilor
  - Implicit: sortare descrescătoare
- Clase predefinite:
  - set
  - multiset
  - map
  - multimap

Clasa set

- Cheia coincide cu valoarea elementului
- Valorile trebuie să fie unice
- Implementare ca arbori binari de căutare
- Elementele din container nu pot fi modificate
  - Pot fi eliminate
- Elementele sînt sortate pe baza unui comparator
- #include <set>

Clasa set

- Numărul de elemente cu o valoare dată
  - count(val)
- Returnarea unui iterator la valoarea cătată
  - find(val)
- Operații/Informare
  - insert(), erase(), swap(), clear(), size(), emtpy()

Clasa set

```
set<string> setNume;

setNume.insert("mihai");
setNume.insert("vlad");
setNume.insert("sorin");
setNume.insert("alina");
setNume.insert("oana");

set<string>::iterator sit;
```

```
for(sit = setNume.begin(); sit != setNume.end(); sit++)
{  cout<<*sit<<endl;}

sit = setNume.find("oana");

if (sit != setNume.end())
{ cout<<"Elem. există"<<endl; }
else
{ cout<<" Elem. nu există "<<endl; }
```

Clasa multiset

- Asemănătoare clasei set
  - Poate conține valori duplicate
- #include <set>
- Metodele din clasa set

Clasa multiset

```
multiset<string> msetNume;

msetNume.insert("mihai");
msetNume.insert("vlad");
msetNume.insert("oana");
msetNume.insert("alina");
msetNume.insert("oana");

multiset<string>::iterator msit;
```

```
msit = msetNume.find("oana");

if (msit != msetNume.end())
{
    cout<<"Apare de "<<
          msetNume.count("oana")<< " ori" << endl;
}
else
{
          cout<<"Nu exista"<<endl;
}
```

Clasa map

- Cheia este distinctă de valoarea elementului
- Perechi (cheie, valoare)
  - pair<tip_cheie, tip_valoare>
- Valorile cheilor nu pot fi duplicate
- Referirea elementelor
  - Operatorul [](tip_cheie) – get/set
- #include <map>

Clasa map

```
map<int, string> angajati;

angajati[100] = "ion vasile";
angajati[102] = "ion ion";
angajati[103] = "vasile vasile";
angajati[101] = "ion ion vasile";

map<int, string>::iterator mit;
mit = angajati.find(101);

if (mit != angajati.end())
{
cout<<"Valoarea pentru cheia: "<<mit->first<<" este "<<mit->second<<endl;
}
```

Clasa map

```
map<int, string> angajati;
angajati.insert(make_pair(100, "ion vasile"));
angajati.insert(make_pair(102, "ion ion"));
angajati.insert(make_pair(103, "vasile vasile"));
angajati.insert(make_pair(101, "ion ion vasile"));
```

```
string nume = angajati[cod];

if (!nume.empty())
{
cout<<"Valoarea pentru cheia "
<<cod<<" este "<<nume<<endl;
}
```

Clasa multimap

- Asemănătoare clasei map
- Poate conține valori duplicate
- Nu include operatorul []
- Pentru obținerea tuturor valorilor cu aceeași cheie se utilizează
  - **equal_range**()
- #include <map>

Clasa multimap

```
multimap<int, string> note;

note.insert(pair<int, string>(10, "ion vasile"));
note.insert(pair<int, string>(10, "ion ion"));
note.insert(pair<int, string>(5, "ion ion vasile"));
note.insert(pair<int, string>(4, "vasile vasile"));
```

```
pair<multimap<int,string>::iterator,
        multimap<int, string>::iterator> rez;

rez = note.equal_range(10);

multimap<int,string>::iterator mmit;

for(mmit = rez.first; mmit != rez.second; mmit++)
    cout << ' ' << mmit->second;

cout <<endl;
```

Containere

	Inserare la început	**Inserare la sfîrșit**	**Inserare în interior**	**Ștergere la început**	**Ștergere la sfîrșit**	**Ștergere din interior**	**Regăsire după poziție**	**Regăsire după cheie**
vector	-	**+**	-	-	**+**	-	**+**	-
deque	**+**	**+**	-	**+**	**+**	-	**+**	-
list	**+**	**+**	**+**	**+**	**+**	**+**	-	-
map							-	**+**
multimap							-	**+**
set							-	**+**
multiset							-	**+**

Containere adaptive

- Adaugă funcționalități containerelor secvențiale
- Nu sînt parcurse cu ajutorul iteratorilor
- Se bazează pe un container secvențial
  - Trebuie selectat inițial
- Metode comune
  - empty()
  - size()
  - swap()
- Clase predefinite:
  - stack
  - queue
  - prority_queue

Clasa stack

- LIFO
- Poate fi adaptat pentru containerele
  - vector
  - list
  - deque (implicit)
- Metode
  - top --> back()
  - push() --> push_back()
  - pop() --> pop_back()

Clasa stack

```
stack<int>  stiva;

//stack<int, vector<int> >  stiva;

stiva.push(10);
stiva.push(20);
stiva.push(30);
```

```
while (!stiva.empty())
{
        cout<<stiva.top()<<endl;
        stiva.pop();
}
```

Clasa queue

- FIFO
- Poate fi implementat pentru containerele:
  - list
  - deque (implicit)
- Metode:
  - front()
  - back()
  - push() --> push_back()
  - pop() -->pop_front()

priority_queue

- Elementele sînt sortate pe baza unui comparator
- Poate fi implementat pentru containerele:
  - vector (implicit)
  - deque
- Metode:
  - top   --> back()
  - push() --> push_back()
  - pop()   --> pop_back()

Iteratori (2)

- Există o serie de iteratori predefiniți
- istream_iterator
  - Formatează intrarea provenită de la un obiect de tip T dintr-un flux de intrare.
  - Permit modificarea elementului pe care îl referă.
- ostream_iterator
  - Realizează legătura unui iterator cu un flux de ieșire
  - Permit doar consultarea elementelor referite
- reverse_iterator
- insert_iteratori

Algoritmi

- Funcții globale
- Fișiere antet
  - <algorithm>
  - <numeric>
- Parametri
  - Iteratori
  - Predicate unare si binare
  - Functori
  - Alte funcții

Algoritmi

- Algoritmi care modifică ordinea elementelor în container (*modifying sequence operations*)
  - **copy**(), **replace**(), **transform**(), **remove**() etc.
- Algoritmi care nu modifică ordinea elementelor în container (*non-modifying sequence operations*):
  - **for_each**(), **find**(), **count**(), **equal**() etc.
- Algoritmi de sortare (*sorting*), căutare binară (*binary search*), partiționare (*partitioning*), operații pe mulțimi (*set operations*) și heap (*heap operațions*):
  - **sort**(), **equal_range**(), **binary_search**(), **partition**(), **includes**(), **merge**(), **set_difference**(), **make_heap**(), **sort_heap**() etc.
- Algoritmi generali pentru operaţii numerice (*numeric*, *minimum/maximum operations*)
  - **min**(), **min_element**(), **max**(), **max_element**(), **accumulate**(), **inner_product**() etc.