PROGRAMAREA ORIENTATĂ OBIECT C++

Conf.univ.dr. Ana Cristina DĂSCĂLESCU

Universitatea Titu Maiorescu

➢STL

- Este o bibliotecă de șabloane utilă pentru crearea și managementul structurilor dinamice de date (tablou, liste, mulțime, tabelă de asocieri etc).

- STL are o arhitectură bazată pe următoarele componente:

  - Containere
  - Algoritmi polimorfici
  - Iteratori

- Programele dezvoltate folosind STL beneficiază de o viteză de dezvoltare și o viteză de executare sporite, fiind mai eficiente, mai robuste, mai portabile și mai ușor de modificat.

➢Containere

- Un *container* este un obiect care grupează mai multe elemente într-o structura unitară.

- Intern, elementele dintr-un container se află într-o relație specifică unei structuri de date (liniară, asociativă, arborescentă etc.), astfel încât asupra lor se pot efectua operații de căutare, adăugare, modificare, ștergere, parcurgere etc.

- În STL exită trei tipuri de containere:
  - *containere secvenţiale* (vector, listă, coadă cu două capete – deque)
  - *containere asociative* (mulţime, relaţie)
  - *containere adaptor*(queue, stack, priority queue)

- Toate template-urile sunt definite în namespace-ul standard și au fișiere-antet de forma <vector>, <stack>, etc

➢ Containerul asociativ de tip mulțime <set>

- Șablonul <set> modelează o colecție de elemente care nu conțin duplicate, respectiv o colecție de tip mulțime.
- Fiecare element din container este o cheie unică!!!
- Permite stocarea și regăsirea rapidă a elementelor.
- Ordinea elementelor nu este dată de ordinea înserarii lor în container, ci de către o funcție comparator!!!
- Containerul asociativ set este implementat intern prin arbori binari de căutare

➢ Containerul asociativ <map>

- Șablonul <map> modelează o colecție de elemente de tip <cheie, valoare>
- Fiecare element din container este o cheie unică!!!
- Se poate utliza atribuirea map[ch]=val
- Permite stocarea și regăsirea rapidă a elementelor (O(logn)).
- Ordinea elementelor nu este dată de ordinea înserarii lor în container, ci de către o funcție comparator!!!
- Containerul asociativ map este implementat intern prin arbori binari de căutare echilibrați

- Containerul map este definit prin două tipuri generice, unul pentru cheie, respectiv unul pentru valoare

- **Exemplu:** pentru a modela o agendă telefonică, containerul contine perechi de tip <string, int>

- Un element din containerul map se definește prin intermediul clasei pair

```
pair<string, int>
```

- Accesarea unui element se realizează prin apelul metodelor:
  `element.first()` - retunnează valoarea cheii
  `element.second()` – returnează valoarea asociată cheii

➢ Operații specifice șablonului <map>

Constructor	Efect	Complexitate
map<tip_ch,tip_val,comp_ch>r	crează o relaţie vidă; sortarea elementelor după cheie se face cu **comp_ch**	**O(1)**

Accesor	Efect	Complexitate
r[ch]	întoarce valoarea accesată prin cheie	**O(log n)**
r.find(ch)	întoarce un iterator la perechea cheie-valoare (**r.end()**, dacă nu găseşte cheia în **r**)	**O(log n)**
r.size()	întoarce numărul curent de elemente	**O(1)**
r.empty()	întoarce **true** dacă relaţia este vidă	**O(1)**
r.begin()	întoarce un iterator la prima pereche din **r**	**O(1)**
r.end()	întoarce un iterator la ultima pereche din **r**	**O(1)**

Modificator	Efect	Complexitate
r[ch]=val	memorează perechea cheie-valoare în **r**	**O(log n)**
m.insert(pereche)	are acelaşi efect	**O(log n)**

➢ Iteratori

- Pentru a parcurge elementele unui container sunt necesare următoarele informații:
  - adresa primului element
  - modalitatea de a obține următorul element celui curent
  - adresa ultimului element

- O soluție unitră pentru a parcurge elementele unui container este dată de **iteratori**

- Rolul principal al iteratorilor este de a accesa elementele unui container, indiferent de tipul acestuia!!!

- Iteratorul este un obiect care permite idirectarea elementelor dintr-un container (pointer către un obiect)

- Tipuri de iteratori
  - iterator - permite modificarea elementelor pe masură ce acestea sunt accesate
  - const_iterator – nu permite modificarea elementelor indirectate

- Definirea iteratorilor de către programator

```
container<T>:: iterator it;
container<T>::const_iterator cit;
container<T>::reverse_iterator rit;
```

- Definirea iteratorilor prin metodele specifice containerului

```
begin(), rbegin()
end(), rend()
```

➢ Iteratori predefiniți

- STL furnizează iteratori predefiniți în headerul **<iterator>**

1. ostream_iterator: conectează un iterator la un flux de ieșire

- pentru a defini un iterator de tip `ostream_iterator`, se precizează tipul elementelor din container, alături de tipul fluxului de ieșire

```
ostream_iterator<int> os_it(cout, " ")
```

2. istream_iterator: conectează un iterator la un flux de intrare

- pentru a defini un iterator de tip `istream_iterator`, se precizează tipul elementelor din container, alături de tipul fluxului de intrare

```
istream_iterator<int> is_it(cin)
```

Algoritmi polimorfici

- Sunt funcții independente care implementează operații specifice colecțiilor de date
- Algoritmii sunt generici, respectiv nu depind de tipul de date
- Sunt definiți în antetul `<algorithm>`

Algoritmul sort

- definește o funcție care sortează elementele unui container
- este o implementare a lui `Quick Sort (O(nlog (n))`
- criteriul de sortare este definit printr-o funcție comparator ce este transmisă ca argument al funcției sort
- funcția comparator primește doua argumente de tipul elementelor aflate în container și returnează o valoare de tipul `bool`

```
qsort(container.begin(),container.end(), comparator);
```

- Exemplu

```
bool cmpPersoana(Persoana ob1, Persoana ob2)
{
     return ob1.getVarsta()<ob2.getVarsta();
}

int main()
{
    Vector<Persoana> vp;
    vp.insert(p1);
    vp.insert(p2);
    sort(v.begin(), v.end(), cmpPersoana);
}
```

- Algoritmul **find**
  - Caută un element într-o colecție și returnează un interator ce referă elemental căutat (prima apariție)
  - Dacă elementul nu se află în colecție, atunci se returnează un iterator ce marchează sfârșitul colecției)

```
vector<int> vec { 10, 20, 30, 40 };
int ser = 30;
it = find (vec.begin(), vec.end(), ser);
    if (it != vec.end())
    {
        cout << "Element " << ser <<" found at position : " ;
        cout << it - vec.begin();
    }
    else
        std::cout << "Element not found";
```

- **for_each**
  - este o funcție care prelucrează elementele unui container, printr-o funcție specificată ca parametru

Exemplu:

```
void afis(int k){cout<<k<<" ";}
int main() {
  vector<int> vi;
  for(int i=0; i<10; i++)
      vi.push_back(i+1);

    for_each(vi.begin(), vi.end(), afis);
    return 0;
}
```

- Modificarea elementelor și salvarea acestor a într-un container

```
transform(sursa.begin(),   sursa.end(),   rezultat.begin(),
regulaTransformare)
```

- Determinare valoare minima/maximă

```
min_element(v.begin(), v. end())
max_element(v.begin(), v. end())
```

- Numărul de apariții ale unui valori

```
count(v.begin(), v. end(), val_cautata)
count_if(count(v.begin(), v. end(), funcCriteriu)
```