Osnovi računarskih sistema

Beleške sa vežbi

Smer *Računarstvo i informatika*
Matematički fakultet, Beograd

Jelena Tomašević

December 12, 2005

Sadržaj

1 Vežbe—— 12.12.2005 **2**
1.1 Apstraktni tipovi kontejnera 2
1.1.1 Iteratori 2
1.1.2 Vektori 4
1.1.3 Liste 7
1.2 Skupovi 9
1.3 Katalozi 10

Čas 1

Vežbe—— 12.12.2005

1.1 Apstraktni tipovi kontejnera

Razlikujemo **sekvencijalne** i **asocijativne** kontejnere.
Sekvencijalni kontejner sadrži uređenu kolekciju elemenata nekog tipa. Dva osnovna sekvencijalna kontejnera koja možemo izdvojiti su **vector** i **list**.
Asocijativni kontejneri su karakteristični po tome što efikasno vrše proveru prisustva kao i izdvajanje pojedinih elemenata. Njihovi elementi su sortirani po ključu. Primer su **map** (katalog) i **set**(skup).

1.1.1 Iteratori

Pod iteratorom podrazumevamo opšti metod pomoću koga pristupamo svakom elementu unutar bilo kog tipa kontejnera. To je pokazivač na element u kontejneru. Npr ako je iter iterator onda ++iter pomera iterator unapred tako da adresira sledeći element u kontejneru a *iter vraća kao rezultat element u kontejneru na koji pokazuje iter.
Za svaki tip kontejnera možemo reći da podržava sledeće f-je:

begin() - f-ja koja kao rezultat vraća iterator koji pokazuje na prvi element u kontejneru.

end() - f-ja koja kao rezultat vraća iterator koji pokazuje na element za 1 iza poslednjeg u kontejneru.

Iterator je ime tipa koji je definisan u kontejneru. Na primer, za klasu vektor, sintaksa je:

```
vector<string>::iterator
```

Osim tipa iterator, u okviru svakog kontejnera je definisan i tip const iterator koji je neophodan za pristupanje elementima konstantnih kontejnera. Ovakav tip iteratora dozvoljava samo čitanje elemenata kontejnera. Npr.

```
const vector<int> *pvec;
vector<int>::const_iterator c_iter_begin = pvec->begin();
```

```
vector<int>::const_iterator c_iter_end = pvec->end();
```

Korišćenjem iteratora možemo da pristupamo npr. srednjem elementu vektora

```
// vector<int> vec;
vector<int>::iterator iter = vec.begin()+vec.size()/2;
```

možemo pristupati svakom drugom elementu

```
iter += 2;
```

i tako dalje. Treba napomenuti da ovakva aritmetika iteratora funkcioniše samo kod vektora jer se kod njega elementi čuvaju u povezanoj memoriji.

Primer 1

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

using namespace std;

int main()
{
    vector<int> niz;
    for( int i=0; i<20; i++ )
        niz.push_back( i );

    // I nacin
    for( int i=0; i<niz.size(); i++ )
        cout << niz[i] << ' ';
    cout << endl;
    //0 1 2 3 4 ... 19

    // II nacin
    int* p = &(niz[0]);
    for( int i=0; i<niz.size(); i++ )
        cout << (*p++) << ' ';
    cout << endl;
    //0 1 2 3 4 ... 19

    // III nacin
    int* poc = &(niz[0]);
    int* kraj = poc + niz.size();
    for( int* p = poc; p!=kraj; p++ )
        cout << (*p) << ' ';
    cout << endl;
    //0 1 2 3 4 ... 19

    // IV nacin
    vector<int>::iterator
        b = niz.begin(),
        e = niz.end();
    for( vector<int>::iterator i=b; i!=e; i++ )
```

```
        cout << (*i) << ' ';
    cout << endl;
    //0 1 2 3 4 ... 19

    return 0;
}
```

1.1.2 Vektori

Pod vektorom podrazumevamo susedne oblasti memorije u kojima se svaki element čuva određenim redosledom.

Osnovne karakteristike vektora su:

1. Nasumični pristup elementima vektora (npr. pristup 5-om pa 17-om elementu i td.) je veoma efikasan (predstavlja fiksni pomeraj u odnosu na početak vektora).
2. Umetanje elementa bilo gde osim na kraj vektora nije efikasno jer bi zahtevalo premeštanje svih elemenata desno od umetnutog za jedno mesto udesno.
3. Brisanje elementa sa bilo kog mesta osim sa kraja nije efikasno jer bi zahtevalo premeštanje svih elemenata desno od izbrisanog za jedno mesto ulevo.

Kako vektor raste?

Da bi vektor dinamički rastao, on mora da obezbedi dodatnu memoriju za čuvanje nove sekvence, da redom iskopira elemente stare sekvence, i da oslobodi memoriju koju je zauzimala stara sekvenca. Ako bi se vektor povećavao posle svakog umetanja to bi bilo neefikasno (pogotovu ako su elementi vektora objekti klasa pa je pri kopiranju za svaki element potrebno pozvati konstruktor kopije a pri brisanju destruktor za taj element). Zato, kad se javi potreba za povećanjem vektora , obezbeđuje se dodatni kapacitet memorije koji prevazilazi trenutne potrebe vektora i čuva se u rezervi. Količina tog dodatnog kapaciteta definisana je kroz implementaciju.

Neke od f-ja koje su definisane u klasi vektor su:

1. **size()** - vraća kao vrednost broj elemenata u vektoru.
2. **resize(x)** - eksplicitno vrši promenu veličine vektora na x.(Treba praviti razliku između veličine i kapaciteta vektora!)
3. **push_back(e)** - vrši dodavanje elementa e odgovarajućeg tipa na kraj vektora.
4. **back(e)** - vraća kao vrednost poslednji element u vektoru.
5. **pop_back()** - vraća kao vrednost poslednji element u vektoru i briše ga.

6. **capacity()** - vraća kao vrednost kapacitet tj. broj elemenata koje je moguće smestiti u vektor.

7. **reserve(x)** - postavlja kapacitet vektora na vrednost x.

8. **empty()** - vraća true ako je vektor prazan, inače false.

9. **insert(iter, e)** - umeće elemenat e na mesto na koje pokazuje iterator iter u vektoru.
   insert(iter, iter1, iter2) - umeću se svi elementi vektora počev od onog na koji pokazuje iterator iter1(uključujući i taj element) do elementa na koji pokazuje iterator iter2(isključujući taj element), na mesto u vektoru na koje pokazuje iterator iter. Vektor čiji se elementi umeću i vektor u koji se elementi umeću ne moraju biti isti.

10. **erase(iter)** - briše se element na koji pokazuje iter.
    erase(iter1, iter2) - brišu se svi elementi u vektoru počev od onog na koji pokazuje iter1(uključujući i taj element) do onog na koji pokazuje iter2(isključujući taj element).

Da bismo mogli da definišemo ili koristimo kontejner potrebno je da uključimo odgovarajuću datoteku zaglavlja.

Primer 2

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

using namespace std;

int main()
{
    // podrazumevani konstruktor pravi prazan niz
    vector<int> niz;
    cout << niz.size() << endl;
    // 0

    // ako navedemo celobrojni parametar, to je velicina niza
    vector<int> niz2( 20 );
    cout << niz2.size() << endl;
    // 20

    // elementima vektora pristupamo pomocu operatora[]
    // !!! KOJI NE PROVERAVA DA LI NIZ IMA DOVOLJNO ELEMENATA !!!
    for( int i=0; i<20; i++ )
        niz2[i] = i;

    // promenu velicine niza mozemo izvoditi eksplictno...
    niz.resize( 50 );
    niz2.resize( 10 );

    // ...ili u koracima za po jedan
```

```
    cout << niz.size() << endl;
    // 50
    niz.push_back( 25 );
    cout << niz.size() << ' ' << niz.back() << endl;
    //51 25
    niz.pop_back();
    cout << niz.size() << endl;
    // 50

    // metodi za rad sa alociranim prostorom
    cout << niz.capacity() << endl;
    //100
    niz.reserve(200);
    cout << niz.capacity() << endl;
    //200
    cout << niz.size() << endl;
    //50

    cout << "Idemo dalje..." << endl;

    return 0;
}
```

Primer 3

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

using namespace std;

int main()
{
    vector<int> niz;
    for( int i=0; i<20; i++ )
        niz.push_back(i);
    for( int i=0; i<niz.size(); i++ )
        cout << niz[i] << ' ';
    cout << endl;
    // 0 1 ... 19

    niz.insert( niz.begin() + 5, 100 );
    for( int i=0; i<niz.size(); i++ )
        cout << niz[i] << ' ';
    cout << endl;
    //0 1 2 3 4 100 5 6 ... 19

    niz.erase( niz.begin() + 5 );
    for( int i=0; i<niz.size(); i++ )
        cout << niz[i] << ' ';
```

```
    cout << endl;
    //0 1 2 ... 19

    niz.insert( niz.begin()+5, niz.begin(), niz.begin()+2);
    for( int i=0; i<niz.size(); i++ )
        cout << niz[i] << ' ';
    cout << endl;
    //0 1 2 3 4 0 1 5 6 ... 19

    niz.erase( niz.begin()+5, niz.begin()+12 );
    for( int i=0; i<niz.size(); i++ )
        cout << niz[i] << ' ';
    cout << endl;
    //0 1 2 3 4 10 11 12 ... 19

    return 0;
}
```

1.1.3 Liste

Lista predstavlja nesusedne oblasti memorije koje su dvostruko povezane parom pokazivača koji pokazuju na prethodni i sledeći element omogućavajući pri tom istovremeno pristupanje elementima i unapred i unazad.

Osnovne karakteristike liste su:

1. Nasumični pristup elementima liste nije efikasan. Naime, da bi se pristupilo nekom elementu neophodno je pristupiti svim prethodnim elementima.
2. Umetanje i brisanje elementa na bilo kom mestu u listi je efikasno.Potrebno je samo premestiti pokazivače ali elemente nije potrebno dirati.
3. Potrebna je dodatna memorija za po dva pokazivača za svaki element liste.

Vektor ili lista?

Pri izboru tipa sekvencijalnog kontejnera postoji nekoliko kriterijuma:

1. Ako se zahteva nasumični pristup elementima, vektor je bolji u odnosu na listu
2. Ako je unapred poznat broj elemenata koje treba sačuvati, opet je poželjnije izabrati vektor.
3. Ako je potrebno često umetati i brisati elemente na mestima različitim od kraja, bolji izbor je lista.
4. Ukoliko je potrebno nasumično pristupati elementima ali i nasumično ih brisati i umetati, vrši se procena u odnosu na cenu nasumičnog pristupa kroz cenu nasumičnog umetanja/brisanja.

Neke od f-ja koje su definisane u klasi list su:

1. **push back(e)** - vrši dodavanje elementa e odgovarajućeg tipa na kraj liste.
2. **push front(e)** - vrši dodavanje elementa e odgovarajućeg tipa na početak liste.
3. **insert(iter, e)** - vrši se umetanje elementa e na mesto u listi na koje pokazuje iterator iter.
4. **erase(iter)** - vrši se brisanje elementa liste na koji pokazuje iterator iter.

Standardna biblioteka obezbeđuje i mnoštvo operacija koje se mogu primeniti nad vektorima i listama a koje nisu obezbeđene kao f-je članice šablona klase vektor ili lista, već kao nezavisni skup generičkih algoritama. Jedan od njih je i algoritam pretraživanja
find(iter1, iter2, e) - vraća kao vrednost iterator koji pokazuje na element e u listi(vektoru) ukoliko ga je tamo pronasao, pretražívajući listu(vektor) od elementa na koji pokazuje iter1(uključujući i taj element) do elementa na koji pokazuje iter2(isključujući taj element). Ukoliko element e nije pronađen među pretraženim elementima, vraća se kao vrednost iterator iter2.
Takođe su obezbeđeni i algoritmi sortiranja, brisanja, numerički algoritmi, relacioni i mnogi drugi.
Da bi ti algoritmi mogli da se koriste neophodno je uključiti odgovarajuće zaglavlje.

```
#include <algorithm>
```

Primer 4

```
#include <algorithm>
#include <list>
#include <iostream>

using namespace std;

int main()
{
    list<int> lista;
    for( int i=0; i<20; i++ )
        lista.push_back(i);
    for( list<int>::iterator i=lista.begin(); i!=lista.end(); i++ )
        cout << (*i) << ' ';
    cout << endl;
    //0 1 2 ... 19

    list<int>::iterator i = lista.begin();
    // Da bi pristupili nekom elementu liste
    // neophodno je da pristupimo i svim prethodnim elementima.
    i++; i++; i++; i++; i++;
    lista.insert( i, 100 );
    for( list<int>::iterator i=lista.begin(); i!=lista.end(); i++ )
```

```
        cout << (*i) << ' ';
    cout << endl;
    // 0 1 2 3 4 100 5 6 ... 19

    lista.erase( i ); // Brise element na koji pokazuje i.
    for( list<int>::iterator i=lista.begin(); i!=lista.end(); i++ )
        cout << (*i) << ' ';
    cout << endl;
    // 0 1 2 3 4 100 6 7 ... 19

    for( int i=0; i<10; i++ )
        lista.push_front(i);
    for( list<int>::iterator i=lista.begin(); i!=lista.end(); i++ )
        cout << (*i) << ' ';
    cout << endl;
    // 9 8 7 6 5 4 3 2 1 0 0 1 2 3 4 100 6 7 8 ... 19

    list<int>::iterator f = find( lista.begin(), lista.end(), 100 );
    if( f != lista.end() ){
        for( int i=0; i<10 && f!=lista.end(); i++, f++ )
            cout << *f << ' ';
        cout << endl;
        //100 6 7 8 9 10 11 12 13 14
        }

    return 0;
}
```

1.2 Skupovi

Skup je jedan od osnovnih tipova asocijativnog kontejnera. On se sastoji iz ključnih vrednosti i vrši efikasnu proveru prisustva nekog elementa.
Operacija koja se najčešće koristi sa skupovima je:
count(e) - vraća kao rezultat 1 ako se element e nalazi u vektoru, inače vraća 0.

Primer 5

```
#include <set>
#include <iostream>

using namespace std;

int main()
{

    set<char> znaci;
    char* s="neki znaci";
```

```
    for( char* p=s; *p; p++ )
        znaci.insert( *p );

    for( char c='a'; c<='z'; c++ )
        cout << c << ' ' << znaci.count(c) << endl;
        // 1 je za slova a c e i k n z

    for( char* p=s+5; *p; p++ )
        znaci.erase( *p );

    for( char c='a'; c<='z'; c++ )
        cout << c << ' ' << znaci.count(c) << endl;
        // 1 je za slova k e
    cout << znaci.size() << endl;
    // 3 (jer se cuva i blanko)

    return 0;
}
```

1.3 Katalozi

Katalog pretstavlja jedan od osnovnih tipova asocijativnog kontejnera. Njegovi elementi su parovi koji su sastavljeni od ključa i vrednosti. Ključ se koristi za pretraživanje a vrednost sadrži neki podatak koji želimo da koristimo. Primer je telefonski imenik koji je odlično podržan katalogom: ime pojedinca je ključ a njemu odgovarajući telefonski broj je vrednost.

Primer 6

```
#include <string>
#include <map>
#include <iostream>

using namespace std;

int main()
{
    map<string,int> ocene;

    // I nacin
    pair<string,int> p;
    p.first = "pera";
    p.second = 8;
    ocene.insert(p);

    // II nacin
    ocene.insert( make_pair( "zika", 6 ));
```

```
    // III nacin
    ocene["persa"] = 9;

    for( map<string,int>::const_iterator i=ocene.begin(); i!=ocene.end(); i++ )
        cout << i->first << " : " << i->second << endl;
        //pera : 8
        //persa : 9
        //zika : 6

    // ovo je neispravno trazenje jer automatski dodaje nepostojece podatke
    char* s[] = { "persa", "zika", "mika", "pera", 0 };
    for( char** p = s; *p; p++ )
        cout << (*p) << " : " << ocene[*p] << endl;
        //persa : 9
        //zika : 6
        //mika : 0
        //pera : 8

    // ovo je ispravno trazenje
    char* s1[] = { "persa", "zika", "mika", "pera", "sasa", 0 };
    for( char** p = s1; *p; p++ ){
        map<string,int>::const_iterator f = ocene.find( *p );
        if( f != ocene.end() )
            cout << (*p) << " : " << f->second << endl;
        else
            cout << (*p) << " jos nije polagao/la" << endl;
        }
        //persa : 9
        //zika : 6
        //mika : 0
        //pera : 8
        //sasa jos nije polagao/la

    return 0;
}
```

I katalog i skup mogu da sadrže samo po jedan primerak svakog ključa. Međutim, postoje i multiskup i multikatalog koji omogućavaju čuvanje i više pojavljivanja ključa.Multikatalog ima primenu npr. u slučaju kada želimo da nekom licu pridružimo više brojeva telefona i da za svaki broj imamo poseban listing.

Primer 7 *Ilustruje se ispisivanje svih elemenata proizvoljne kolekcije koristeći šablonsku funkciju.*

```
#include <vector>
#include <list>
#include <set>
#include <map>
```

```
#include <iostream>

using namespace std;

// u opstem slucaju iteratora imamo:
//  - na kolekciji:
//      begin()
//      end()
//  - na iteratoru:
//      i++ - sledeci
//      *i  - dereferisanje
//      i->x    - ako je element kolekcije struktura...
//              isto kao (*i).x

template<class kolekcija>
void ispisiSveElemente( const kolekcija& k )
{
    class kolekcija::const_iterator
        i = k.begin(),
        e = k.end();
    for( ; i!=e; i++ )
        cout << (*i) << ' ';
    cout << endl;
}

int main()
{
    vector<int> niz;
    for( int i=0; i<20; i++ )
        niz.push_back( i );
    ispisiSveElemente( niz );
    // 0 1 ... 19

    list<float> lista;
    for( float x=0; x<50; x+=1.35 )
        lista.push_back( x );
    ispisiSveElemente( lista );
    // 0 1.35 2.7 ... 49.95

    set<char> znaci;
    char* s="neki znaci";
    for( char* p=s; *p; p++ )
        znaci.insert( *p );
    ispisiSveElemente( znaci );
    //a c e i k n z

    return 0;
}
```