Sveučilište u Zagrebu
PMF – Matematički odjel

Objektno programiranje (C++)

Vježbe 02 – STL

Vinko Petričević

Tip `string`

- `string`
  - niz znakova varijabilne duljine
  - samostalni memory management
  - dovoljno efikasan za generalnu upotrebu
  - mnoge korisne operacije na stringovima
- header datoteka `<string>`
  - `#include <string>`
- tip string nalazi se u namespaceu `std`
  - slideovi pretpostavljaju deklaraciju

```
using std::string;
```

Definicija i inicijalizacija stringova

- Konstruktori:
  - `string s1;`
    - defaultni konstruktor – `s1` je prazan string
  - `string s2(s1);`
    - inicijalizacija stringa `s2` s kopijom od `s1`
  - `string s3("ABC");`
    - inicijalizacija stringa `s3` s kopijom string literala
  - `string s4(n, 'c');`
    - inicijalizacija stringa `s4` s n kopija znaka `'c'`

Definicija i inicijalizacija stringova

- String možemo istovremeno definirati i inicijalizirati na slijedeći način:

```
string s = "xyz";
```

- Pitanje: Što ovakav izraz zapravo predstavlja?
  - `string s("xyz"); // dozvoljena optimizacija`
  - `string s(string("xyz")); // standard`
- Poželjno je uvijek koristiti konstruktor-sintaksu za inicijalizaciju!

Podsjetnik: ulaz i izlaz

- Header datoteka: `<iostream>`
- `std::cout` - objekt klase ostream
- `std::cin` - objekt klase istream
- Primjer:

```
int i;
std::string s;
std::cin >> i;
std::cin >> s;
std::cout << "string: " << s << std::endl
          << "broj: " << i << std::endl;
```

- ignoriraju se vodeće bjeline
- učitava se niz znakova do prve slijedeće bjeline

Čitanje i pisanje stringova

- Primjer:
  - ```
    string s1, s2;
    cin >> s1 >> s2; // pročitaj prvo s1, zatim s2
    cout << s1 << s2 << endl;
    ```
- Čitanje nepoznatog broja stringova:
  - ```
    string word;
    // čitaj sve do EOF-a
    while (cin >> word)
        cout << word << endl;
    return 0;
    ```
- Čitanje cijele linije:
  - ```
    string line;
    // čitaj liniju po liniju sve do EOF-a
    while (getline(cin, line))
        cout << line << endl;
    ```

stanje streama kao logički uvjet

getline() učitava liniju teksta s danog ulaznog streama (sve do prelaska u novi red koji se odbacuje) u dani string

std::istream &std::getline(std::istream&, std::string&);
    ```

Operacije na stringovima

- Duljina stringa dobiva se pomoću funkcije `size()`:

```
string s("OP");
cout << "Duljina od " << s
 << " je " << s.size() << " znakova.";
```

- Napomena: `size()` vraća `string::size_type`
- Provjera da li je string prazan:

```
string s2;
if (st.size() == 0)
    // ok: prazan string
if (s2.empty())
    // ok: prazan string
```

Operacije na stringovima

- Relacijski operatori:
  - ```
    string big = "big", small = "small";
    string s1 = big;  // s1 je kopija od big
    if (big == small) // neistina
        // ...
    if (big <= s1)    // istina
        // ...
    ```
- Pridruživanje – kopiranje jednog stringa u drugi:

```
s1 = small; // kopira string small u s1
```

- Konkatenacija stringova:

```
string s1("hello");
string s2("world");
string s3 = s1 + ", " + s2 + "\n";
s1 += s2;
```
```

Primjer: spajanje stringova

```
string spoji1(string a, string b) {
    return a + "." + b;
}
```

```
string spoji2(string a, string b) {
    string ret(a.size()+1+b.size(),'.');
    ret.replace(0, a.size(), a);
    // copy(b.begin(), b.end(),
         ret.begin()+a.size()+1);
    copy(b.begin(), b.end(), &ret[a.size()+1]);
    return ret;
}
```

Operacije na stringovima

- Napomena: Redefinirani operatori zadržavaju asocijativnost i prioritete!
- Pitanje: Koji od slijedećih izraza su legalni?
  - `s = t + "baz";`
  - `s = "baz" + t;`
  - `s = t + "baz" + "bop";`
  - `s = "baz" + t + "bop";`
  - `s = "baz" + "bop" + t;`

  (u svakoj konkatenaciji mora sudjelovati barem jedan objekt tipa string, pa `"baz" + "bop"` predstavlja grešku)

Operacije na stringovima

- Operator [] – pristup individualnim znakovima u stringu
  - ```
    string str("neki string");
    for (string::size_type i = 0; i != str.size(); ++i)
        cout << str[i] << endl;
    ```
- Operator [] vraća lvalue, tj. `str[i]` je tipa `char&`:
  - ```
    for (string::size_type i = 0; i != str.size(); ++i)
        str[i] = '*';
    ```
  - ipak, ako bi `str` bio definiran kao const objekt:

    ```
    const std::string str;
    ```

    `str[i]` će biti tipa `const char&`, pa tada nije dozvoljeno pridruživanje!

Operacije na stringovima

- Tip `string` i C-stil stringovi:

```
std::string s;
const char *pc = "polje karaktera";
s = pc; // ok
char *str = s; // greska
```

- Funkcija `c_str()` vraća reprezentaciju stringa u C-stilu:

```
char *str = s1.c_str(); // greska
const char *str = s1.c_str(); // ok
```

Operacije na stringovima

- Vraćanje podstringa: `substr(pocetak, duljina)`
  - ```
    string A("Nesto"), B;
    B = A.substr(3, 2); // B="to"
    ```
- Traženje podstringa: find(stoTrazim, gdjePocinjem), vraća mjesto
  - ```
    string A("kokodako");
    int gdje = A.find("ko", 1);
    // trazi "ko" pocevsi od 1.mjesta (ne 0.)
    // gdje=2 jer se "ko" kao podstring
    // prvi put javlja na 2.mjestu
    ```
- Ako `find()` ne uspije naći podstring, vraća `string::npos`
  - ```
    string S("kokoda"), T("kokos");
    int gdje = S.find(T, 0);
    if (gdje == string::npos)
        cout << "nema ga";
    ```
- Brisanje podstringa: `erase(pocetak, koliko)`
  - ```
    string S("nestodrugo");
    S.erase(2, 6); // sad je S="nego"
    ```

vector

- Generički kontejner objekata istog tipa koji predstavlja alternativu C++ poljima
- header datoteka `<vector>`
  - `#include <vector>`
- Slideovi pretpostavljaju deklaraciju

```
using std::vector;
```

- Primjer:

```
#include <vector>
vector<int> a(10);
```

Operacije nad vektorom od 10 `int`-ova korespondiraju operacijama nad poljem od 10 `int`-ova:

```
int a[10];
```

vector

- Primjer korištenja vectora kao polja:

```
const int elem_size = 10;
vector<int> a(elem_size);
int b[elem_size];
// ...
for (int i = 0; i < elem_size; ++i)
    a[i] = b[i];
// ...
if (a.empty())
    // neistina
// ...
for (int i = 0; i < a.size(); ++i)
    cout << a[i] << ' ';
```

vector

- Konstruktori:
  - `vector<T> v1;`
    - defaultni konstruktor – `v1` je prazan vektor (s 0 elemenata)
  - `vector<T> v2(v1);`
    - `v2` sadrži kopije elemenata od `v1` (`v1` i `v2` moraju biti istog tipa)
  - `vector<T> v3(n, i);`
    - `v3` sadrži `n` elemenata, svaki od kojih je inicijaliziran kopijom vrijednosti i
  - `vector<T> v4(n);`
    - `v4` sadrži `n` elemenata, svaki od kojih je defaultno konstruiran

vector

- Napomena: vector kao takav nije tip podatka, već *predložak* za generiranje različitih tipova:
  - `vector<int>`
  - `vector<string>`
  - `vector<vector<double> >`
- Napomena: Kod konstruktora oblika

  ```
  vector<T> v(n);
  ```

  tip `T` mora biti default konstruktibilan:
  - primitivni tip
  - korisnički tip s defaultnim konstruktorom

  (stoga `T` npr. ne može biti tip reference)

Operacije na vektorima

- `v.empty()`
  - vraća `true` ako je `v` prazan; inače vraća `false`
- `v.size()`
  - vraća broj elemenata u vektoru `v`
- `v.clear()`
  - brisanje svih elemenata vektora `v`
- `v[n]`
  - vraća element na poziciji `n` u vektoru `v`
  - povratni tip je `T&` (ili `const T&` ako je vektor konstantan)
- `v1 = v2`
  - pridružuje vektoru `v1` kopije elemenata iz `v2` (tipovi vektora `v1` i `v2` moraju biti identični)

Operacije na vektorima

- `v.push_back(t)`
  - dodaje kopiju od `t` kao novi element na kraj vektora i povećava mu veličinu za 1 (može implicirati alokaciju memorije)
  - amortizirano konstantno vrijeme izvršavanja
- `v.pop_back()`
  - izbacuje element s kraja vektora
- `==`, `!=`, `<`, `<=`, `>` i `>=`
  - svi relacijski operatori definirani tako da vektore uspoređuju leksikografski (analogno kao kod tipa string)

Primjeri

- Čitanje stringova sa standardnog ulaza ubacujući pritom jedan po jedan u vector:

```
vector<string> text;
string word;
while (cin >> word) {
        text.push_back(word);
        // ...
}
```

- Iteriranje kroz elemente pomoću operatora `[]`:

```
cout << "procitane rijeci:\n";
for (int i = 0; i < text.size(); ++i)
    cout << text[i] << ' ';
cout << endl;
```

- Iteriranje kroz elemente pomoću iteratora:

```
cout << "procitane rijeci:\n";
for(vector<string>::iterator it=text.begin();
it != text.end(); ++it)
    cout << *it << ' ';
cout << endl;
```

Iteratori

- Iteratori su tipovi pridruženi svakom kontejnerskom tipu zasebno i namijenjeni su za pristupanje elementima u redoslijedu podržanom od strane kontejnera (npr. slijedno, obrnutim poretkom, itd.)
- Korištenje iteratorskih objekata sintaktički korespondira korištenju pokazivača
  - u C++u pokazivače i shvaćamo kao iteratore na poljima
- Motivacija:
  - ```
    int a[10];
    int *const begin = a;
    int *const end = a + 10;
    for (int *iter = begin; iter != end; ++iter)
        std::cin >> *iter;
    ```
    ```

iterator

- ```
  std::vector<int> v(10);
  for (vector<int>::iterator iter = v.begin();
       iter != v.end(); ++iter)
           std::cin >> *iter;
  ```
  - `v.begin()` – vraća iterator koji pokazuje na početni element kontejnera
  - `v.end()` – vraća iterator koji pokazuje "iza zadnjeg" elementa kontejnera
  - `(v.begin() == v.end())` akko `v.empty()`
  - `++` (inkrement) iteratora pozicionira ga na slijedeći element kontejnera u danom redoslijedu
  - dereferenciranje iteratora vraća referencu na objekt – element kontejnera – na kojeg iterator trenutno pokazuje
  ```

const_iterator

- const_iterator je iterator koji dereferenciran vraća const referencu na pripadni element kontejnera
- koristi se kad ne treba mijenjati elemente
  - ```
    for (vector<int>::const_iterator iter = v.begin();
        iter != v.end(); ++iter)
            std::cout << *iter << endl;
    ```
- const_iterator radi i na konstantnim kontejnerima
  - ```
    const vector<int> cv(42);
    vector<int>::const_iterator iter = cv.begin();
    ```
- const_iterator != const iterator
  - ```
    const vector<int>::iterator it1 = v.begin();
    *it1 = 7; // ok
    ++it1; // greska, jer jer je iterator konstantan
    ```
  - ```
    vector<int>::const_iterator it2 = v.begin();
    // greska, jer je s lijeve strane const int&
    *iter = 8;
    ```

Aritmetika iteratora

- Aritmetika iteratora – aritmetika pokazivača
- `iter + n`
  - vrijednost ovog izraza je novi iterator koji pokazuje na objekt koji je `n` objekata "desno" od *iter
- `iter - n`
  - vrijednost ovog izraza je novi iterator koji pokazuje na objekt koji je `n` objekata "lijevo" od *iter
- `n` je tipa size_type ili difference_type danog kontejnera
  - difference_type je cjelobrojni tip s predznakom i javlja se kao rezultat oduzimanja iteratora (`iter1 - iter2`)
- Vrijedi: `iter1 = iter2 + (iter1 - iter2)`
- Napomena: iterator postaje invalidan nakon promjene strukture kontejnera (npr. nakon poziva push_back(), pop_back(), itd.)

Još konstruktora za vector

- Primjer:

```
#include <vector>
#include <iostream>
using namespace std;

void main() {
  int a[] = {7, 8, 9};

  // konstrukcija vektora pomocu pokazivaca
  vector<int> vi(a, a+3);

  // ili pomocu iteratora
  vector<int> vj(vi.begin(), vi.end());

  for(int i=0;i<vj.size(); ++i)
    cout << vj[i] << " ";
  cout << endl;
}
```

Zadaci

- Zadatak: Napišite program koji čita stringove sa standardnog ulaza ubacujući pritom jedan po jedan u vektor, te nakon unosa EOF (^Z) ispisuje sadržaj dobivenog vektora.
- Zadatak: Napišite program koji čita stringove sa standardnog ulaza sve dok se ne učita EOF, te potom ispisuje koliko se puta pojedini string pojavio na ulazu.

Tipovi apstraktnih spremnika

- Slijedni spremnici (sekvencijalni)
  - služe za čuvanje uređene kolekcije elemenata određenog tipa
  - osnovni tipovi
    - vector ,list, deque
  - adaptirani tipovi: stack, queue, priority_queue
- Asocijativni spremnici
  - pružaju podršku za efikasno pronalaženje elemenata na temelju ključa
  - tipovi: map, multimap, set, multiset

Tipovi apstraktnih spremnika

- Spremnici se razlikuju po načinu pristupa elementima i po "cijeni" operacija nad elementima (čitanje, dodavanje, brisanje)
- Spremnici definiraju relativno mali osnovni broj operacija
  - Dio operacija nude svi tipovi spremnika
  - Dio operacija je specifičan za slijedne tj. asocijativne spremnike
  - Dio operacija je specifičan za konkretan spremnik
- Puno više operacija definiraju biblioteke algoritama [Lippman, Ch.11]

Slijedni spremnici (sekvencijalni)

- služe za čuvanje uređene kolekcije elemenata određenog tipa
- elemente dohvaćamo po poziciji (indeksu), npr. a[i]
- osnovni tipovi:
  - vector – “polje”: brz pristup pojedinom elementu
  - list – vezana lista: brzo ubacivanje i brisanje
  - deque – red sa “dva kraja” (double-ended queue)

Slijedni spremnici

- adaptirani tipovi:
  - stack – stog: LIFO
  - queue – red: FIFO

  } Na temelju deque-a

  - priority_queue – prioritetni red

  } Na temelju vector-a

- Adaptori – prilagođuju slijedni spremnik koji se krije "ispod površine" tako da mu definiraju novo sučelje

Slijedni spremnici

- Zaglavlja:
  - vector – `#include <vector>`
  - list – `#include <list>`
  - deque – `#include <deque>`
  - stack – `#include <stack>`
  - queue – `#include <queue>`
- Definicija spremnika sastoji se od navođenja imena spremnika, te tipa elemenata koje želimo čuvati
  - `vector<string> svec;`
  - `list<int> ilist;`
  - `deque<double> dd;`
  - `queue<float> fq;`
  - `stack<char> cstack;`

Konstrukcija slijednih spremnika

- Inicijalizacija spremnika elementima polja

```
string words[4] = {"abc", "xyz", "foo", "bar"};
vector<string> vwords(words, words+4);
int a[6] = { 0, 1, 2, 3, 4, 5 };
list<int> ilist(a, a+6);
```

- Inicijalizacija spremnika iteratorima

```
vector<int> ivec(ilist.begin(), ilist.end());
list<int> ilist1(ilist.begin(), ilist.end());
// list<int> ilist2(ilist);
```

- Smijemo koristiti i spremnike spremnika

```
vector< list<int> > vli; //vektor liste intova
```

  - Obratite pažnju na razmak između > >
- ...i spremnike spremnikovih spremnika...

```
vector< list< deque< list <char> > > > vldli;
```

Ograničenja na podatke u spremniku

- Tip podataka koji se nalazi u slijednom spremniku mora podržavati pridruživanje (=) i kopiranje
  - Neke operacije nad spremnicima imaju i dodatne zahtjeve
    - Ako tip nije “dovoljno dobar”, moći ćemo napraviti spremnik, ali možda nećemo moći isvršavati sve operacije nad spremnikom
  - Reference ne podržavaju kopiranje, pa ne možemo imati spremnik referenci
  - Ograničenja kod asocijativnih spremnika su još veća
- Svi slijedni spremnici rastu dinamički

Iteratori

- Kako doći do unutarnjih elemenata spremnika?
  - Na listi ne radi `L[3]`
- Neka je `iter` iterator u bilo kojem spremniku
  - `++iter;  iter++;`
    pomiče iterator tako da pokazuje na idući element u spremniku
  - `--iter;  iter--;`
    pomiče iterator tako da pokazuje na idući element u spremniku
  - `*iter;`
    vraća vrijednost elementa na kojeg pokazuje iterator
    - Kao i kod pokazivača vrijedi: `*iter.mem = iter->mem`

Iteratori

- `iter1 == iter2; iter1 != iter2;`
- da li su dva iteratora jednaka tj. različita (jednaki su ako pokazuju na isto mjesto u istom spremniku)
- Jednakost iteratora ne mora značiti da su i sadržaji na koje pokazuju jednaki (`*iter1 == *iter2;`)

- Raspon iteratora (iterator range) se interpretira kao:

  `[pocetak, kraj>`

  - Uključena je lijeva granica, ali ne i desna

Iteratori

- `vector` i `deque` podržavaju dodatne operacije nad iteratorima:
  - `iter + n; iter += n`
    pomiče iterator za n mjesta u “desno”
  - `iter – n; iter –= n;`
    pomiče iterator za n mjesta u “lijevo”
  - `iter1 – iter2;`
    vraća broj `n` takav da je `iter1 + n == iter2`
  - `< <= > >=`
    iter1 je manji od iter2 ako je pozicija iter1 ispred pozicije iter2

Tipovi u spremnicima

- Tipovi definirani u slijednim spremnicima:
  - `size_type` tip koji je dovoljno velik da podnese veličinu bilo kojeg spremnika
  - `iterator`
  - `const_iterator`
  - `reverse_iterator`
  - `const_reverse_iterator`
  - `value_type` tip podataka u spremniku

Iteratori na konstantnim spremnicima

- Za iteriranje po konstantnom spremniku potrebno je koristiti `const_iterator`

```
void even_odd(const vector<int> *pvec,
              vector<int> *pvec_even,
              vector<int> *pvec_odd) {
  vector<int>::const_iterator c_iter=pvec->begin();
  vector<int>::const_iterator c_iter_end=pvec->end();

  for ( ; c_iter != c_iter_end; ++c_iter )
     if (*c_iter % 2)
         pvec_odd->push_back(*c_iter);
     else pvec_even->push_back(*c_iter);
}
```

- `reverse_iterator` koristimo za kretanje po spremniku unazad

Operacije na spremnicima

- Iteratori:
  - `c.begin()` vraća iterator koji adresira prvi element u spremniku
  - `c.end()` vraća iterator koji adresira element iza posljednjeg elementa u spremniku
    - (ne smijemo ga dereferencirati)
  - `c.rbegin()` vraća reverse iterator ("`end() - 1`")
  - `c.rend()` vraća reverse iterator ("`begin() - 1`")
- Ubacivanje elemenata
  - `c.push_back(el)`
  - `c.push_front(el)`
  - `c.insert(p, el)` //ubacuje el ispred pozicije p; vraća iterator na el
  - Operacije ubacivanja mogu uništiti iteratore

Ubacivanje elemenata

- ```
  vector<string> svec;
  list<string> slist;
  string s1("foo");
  slist.insert(slist.begin(), s1);
  svec.insert(svec.begin(), s1);
  ```
- ```
  string s2("bar");
  list<string>::iterator iter;
  iter = find(slist.begin(), slist.end(), s1);
  slist.insert(iter, s2);
  ```

- Poziv metode `push_back()` ekvivalentan je s:

  ```
  // slist.push_back(value);
  slist.insert(slist.end(), value);
  ```

- Možemo koristiti i način `insert(i_gdje, i_poc, i_kraj);`

Ubacivanje elemenata

- push_back()

```
string text_word;
while (cin >> text_word)
      svec.push_back(text_word);
int a[4] = {0, 1, 2, 3};
for (int i = 0; i < 4; ++i)
      ilist.push_back(a[i]);
```

- push_front() // ne radi na vektoru

```
for (int i = 0; i < 4; ++i)
      ilist.push_front(a[i]);
```

Zadaci

- Zadatak: Napišite program koji učitava i stavlja na kraj liste stringove sve dok se ne učita string “kraj”
- Zadatak: Ispišite sadržaj gore dobivene liste
  - Pomoću pop_front() i front() operacija
  - Pomoću iteratora
- Zadatak: ispred svakog stringa S u listi dodajte još onoliko čvorova koliko S ima slova, u svaki od čvorova upišite po jedno slovo od S
  - npr. ako je učitana lista bila (“RP4”, “Jupi”), onda rezultantna lista treba biti (“R”, “P”, “4”, “RP4”, “J”, “u”, “p”, “i”, “Jupi”)
- Zadatak: na kraju premjestite iz liste u vektor sve stringove duljine veće od 1
- DZ: isti zadatak, ali nove čvorove treba dodavati iza učitanih

Iteratori

```
list<string> L(5, “abc”);
list<string>::iterator li;

for (li=L.begin(); li!=L.end(); li++) {
    string element = *li;
    cout << element << “ “;
}
```

Size operacije

- vector se ne povećava sa svakim pojedinim ubacivanjem elementa, nego se prilikom pojedinih povećavanja alocira i još nešto dodatnog prostora
- Terminologija:
  - kapacitet – ukupan broj elemenata koji se mogu nalaziti u spremniku prije opetovanog povećavanja – capacity()
  - veličina – trenutni broj elemenata u spremniku – size()
  - maksimalna veličina – maks. broj elemenata u spremniku – max_size()

Size operacije

- `v.reserve(nova_velicina); // capacity`
  `v.resize(nova_velicina);  // size`
- Reserve i resize mogu uništiti iteratore
- Zadatak: isprobajte kako se povećavaju size i capacity kod vektora, liste i reda prilikom dodavanja elemenata.

```
c<int> ci;
for(int i=0; i<25; ++i) {
   ci.push_back(i*i);
   cout << "size = " << ci.size() ;
   cout << "cap  = " << ci.capacity() ;
 }
```

Pristup elementima spremnika

- `c.back()` vraća referencu na element sa kraja spremnika
- `c.front()` vraća referencu na element sa početka spremnika
  - Nije definirano ako je spremnik prazan
- `c[n]` vraća referencu na n-ti element spremnika
  - Nije definirano ako je n van granica spremnika
  - Samo za vektor i deque
- `c.at(n)` vraća referencu na n-ti element spremnika
  - Ako je indeks van dosega, baca `out_of_range` izuzetak
  - Samo za vektor i deque
- Reserve i resize mogu uništiti iteratore

Pristup elementima spremnika

- Primjer za list (dequeue)
  - `list<int> L; // L=()`

```
L.push_back(5);  // L=(5)
L.push_front(7); // L=(7,5)
L.push_back(3);  // L=(7,5,3)

int a = L.front(); // a = 7
int b = L.back();  // b = 3

L.pop_front();   // L=(5,3)
L.pop_back();    // L=(5)
```

Brisanje elemenata

- `c.erase(p)` briše element na kojeg pokazuje iterator p
  - vraća iterator na element iza obrisanog
- `c.erase(b,e)` briše sve elemente između dva iteratora
- `c.clear()` briše sve elemente spremnika
- `c.pop_back()` briše zadnji element spremnika
- `c.pop_front()` briše prvi element spremnika
  - Samo za listu i deque

- Brisanje elemeneta može uništiti iteratore

Brisanje elemenata

- ```
  string searchValue("FooBar");
  list<string>::iterator iter =
   find(slist.begin(), slist.end(),
  searchValue);
  if (iter != slist.end())
  slist.erase(iter);
  ```
- Možemo obrisati i niz elemenata određen dvama iteratorima [>
  - ```
    list< string >::iterator first, last;
    first = find(slist.begin(), slist.end(),
    val1);
    last = find(slist.begin(), slist.end(),
    val2);
    slist.erase(first, last);
    ```

Brisanje elemenata

- brisanje svih elemenata liste jednakih 5 – `erase`:

```
list<int> L; ... // napuni nekako L
list<int>::iterator li, ltemp;
li = L.begin();
while (li != L.end())
   if (*li == 5) {
     L.erase(li); li++; //opasno!
   }
     else li++;
```

- Pokušali smo povećati iterator koji smo upravo "uništili"
- oprez: slična stvar se može desiti i neopreznim korištenjem naredbi `push_back`, `pop_back` i `insert` na vectoru.

Brisanje elemenata

- Rješenje 3: brisanje svih elemenata liste jednakih 5 – `erase`:

```
list<int> L; ... // napuni nekako L
list<int>::iterator li, ltemp;

li = L.begin();
while (li != L.end()) {
   if (*li == 5) {
      li = L.erase(li);
   }
   else li++;
}
```

- Iskoristili smo činjenicu da erase vraća iterator na element iza obrisanog
- Dobili smo i korektno i brzo rješenje

Iteratori – “opasne” operacije

- Neke operacije nad spremnicima mogu uništiti (invalidirati) iteratore
  - Neke operacije uništavaju samo sve one iteratore koji pokazuju na određeni element (npr. erase)
  - Neke operacije uništavaju i iteratore koji pokazuju na neke druge elemante u istom spremniku (npr. insert)

Spremnici – uspoređivanje

- `c1 == c2; c1 != c2;` jednakost tj. nejednakost
  - c1 i c2 moraju biti istog tipa
- `c1 < c2; c1 <= c2;`
- `c1 > c2; c1 >= c2;`
  - dozvoljeno samo ako se navedene operacije mogu izvršavati na elementima unutar spremnika
  - Uređaj je leksikografski

- Zadatak: Napišite program koji uspoređuje vektor i listu leksikografski (element po element). Sjetite se da kod liste nemate operator[ ].

Pridruživanje i zamjena i uspoređivanje

- kopiranje vektora

```
vector<int> S(10,3), A(5);
A = S;                  // A=(3,3,3,3,3,3,3,3,3,3)
```

- `S==A; //true`
- `S!=A; //false`
- leksikografski uređaj – `<=, <, >=, >`

```
vector<int> S, T;
S.push_back(3); S.push_back(1);
T.push_back(3); T.push_back(5);
if (S < T) {...}  // istina: (3,1)<(3,5)
```

- Zadatak: isprobajte da li možete usporediti dvije “liste vektora integera”.

Kako raste vektor

- vektor se ne povećava sa svakim pojedinim ubacivanjem elementa, nego se prilikom pojedinih povećavanja alocira i još nešto dodatnog prostora
- Terminologija:
  - kapacitet – ukupan broj elemenata koji se mogu nalaziti u spremniku prije opetovanog povećavanja – capacity()
  - veličina – trenutni broj elemenata u spremniku – size()
- Način povećavanja vektora je ovisan o implementaciji biblioteke

Kako raste vektor

- `vector.reserve(nova_velicina); // capacity`
  Reserve može uništiti iteratore
- Zadatak: isprobajte kako se povećavaju size i capacity kod vektora prilikom dodavanja elemenata.

```
c<int> ci;
for(int i=0; i<25; ++i) {
   ci.push_back(i*i);
   cout << “size = ” << ci.size() ;
   cout << “cap  = ” << ci.capacity() ;
 }
```

- Napomena: capacity i reserve su specifični za vektor i ne pojavljuju se kod dugih spremnika

Primjer

```
vector<int> S(2);   // S=(0,0)
S.push_back(3);     // S=(0,0,3)
S.pop_back();       // S=(0,0)
S.push_back(4);     // S=(0,0,4)
S.resize(5);        // S=(0,0,4,0,0);
S.reserve(10);      // S=(0,0,4,0,0);
cout<<S.front();    // S[0]
S.back()=5;         // S[S.size()-1]
S.clear();          // S=();
S.empty();          // da li je S
prazan?
```

Izbor slijednog spremnika

- Spremnik biramo obzirom na njegove karakteristike ubacivanja, pretraživanja i brisanja elemenata
- Neki kriteriji odabira pogodnog slijednog spremnika
  - ukoliko trebamo direktan pristup elementima iz spremnika: `vector` ili `deque`
  - ukoliko unaprijed znamo broj elemenata koje trebamo spremiti: `vector` ili `deque`
  - ukoliko trebamo ubacivati ili brisati elemente na pozicijama različitim od krajeva spremnika: `list`
  - ukoliko ne trebamo ubacivati ili brisati elemente na početnom kraju spremnika: `vector`

Stack, queue

- stack

```
#include <stack>
...
stack<int> S;
S.push(3);
S.push(5);

int a = S.top();
S.pop();

if (S.empty())
  { ... }

int zz = S.size();
```

- queue

```
#include <queue>
...
queue<string> Q;
Q.push("abc");
Q.push("xy");

string a = Q.front();
Q.pop();

if (Q.empty())
  { ... }

int zz = Q.size();
```

Još elemenata C++ standardne biblioteke

- `bitset`
  - niz bitova fiksne duljine
- `complex`
  - kompleksni broj
- `pair`
  - uređeni par dva objekta

complex

- Omogućava korištenje kompleksnih brojeva
  - `complex<double> z1(0, 7); // 0 + 7*i`
  - `complex<float> z2(3); // 3 + 0*i`
  - `complex<long double> zero; // 0 + 0*i`
- Može i ovo:
  - `complex<int> ci(2, 2);`
  - `complex<string> sc("abc", "xyz");`
- Potrebno je uključiti zaglavlje `<complex>`
  - `#include <complex>`
- `complex` je predložak (isto kao i `vector`)

complex

- Zadatak
  - Isprobajte `cin` i `cout` na complex-u
  - Isprobajte aritmetičke operacije (`+, -, *, /`) na kompleksnim brojevima
  - Isprobajte uspoređivanje (`==, !=, <, >, <=`, ...)
  - Napišite funkciju koja računa apsolutnu vrijednost kompleksnog broja
    - Što prima takva funkcija?
    - Koji je povratni tip funkcije?
      - Pravo rješenje ćemo vidjeti na jednoj od sljedećih vježbi

bitset

- Niz bitova fiksne duljine
  - `bitset<8> bajt; // 8 bitova`
  - `bitset<13> b_ul(19); // 13 bitova sa bin.zapisom 19`
    - `// ”višak” bitova u bitsetu se napuni nulama`
  - `bitset<19> b_s(string(“100110”));`
  - `bitset<2> b_ul2(6);`
    - `// uzimamo dva “desna” bita iz zapisa od 6 (“110”)`
- Probajte:
  - `bitset<7> b_s2(“100110”);`
- Potrebno je uključiti zaglavlje `<bitset>`
  - `#include <bitset>`
- `bitset` je također predložak (isto kao i `vector`)

bitset

- Pristup pojedinom bitu u bitsetu
  - `bitset<8> bajt(7); // “00000111”`
  - `cout << bajt[2];`
- Isprobajte slijedeće:

```
bajt[2] = 0;
cout << bajt;
bajt[2] = 5; // sve sto nije 0 je 1
cout << bajt;
```

bitset

- Operacije na `bitset` -u
  - `b.any();`
  - `b.none();`
  - `b.count();`
  - `b.size();`
  - `b[pos];`
  - `b.test(pos);`
  - `b.set();`
  - `b.set(pos);`
  - `b.reset();`
  - `b.reset(pos);`
  - `b.flip();`
  - `b.flip(pos);`
  - `b.to_ulong();`
  - `os << b;`

bitset

- Logičko “i”
  - `bitset<8> b1(7); bitset<8> b2(61);`
  - `bitset<8> r;`
  - `for (size_t i = 0; i != b1.size(); ++i)`
    - `r[i] = b1[i] && b2[i]; // 1. varijanta`
    - `if (b1[i] && b2[i]) r.set(i); // 2. varijanta`
- Jednostavnije:
  - `cout << (b1 & b2);`
- `size_t` je tip definiran u `cstddef` zaglavlju (C++ verzija od `stddef.h`)
  - Radi se o unsigned tipu za kojeg je garantirano da je dovoljno velik da može sadržavati veličinu bilo kojeg objekta u memoriji

bitset

- Zadatak:
  - Pretpostavimo da igramo Loto 6/45. Izvučene brojeve predstavljamo bitsetom duljine 45 (46 ukoliko ne računamo 0). Napišite funkciju koja simulira izvlačenje brojeva (podsjetnik: `rand()` & `srand(time(0))`) i sprema ih u globalni bitset. Napišite i funkciju koja popunjava listić – niz od 10 kombinacija koje učitavamo sa tipkovnice. Za kraj napišite i funkciju koja računa koliko ste brojeva pogodili vašim listićem.

pair

- Omogućava stvaranje uređenih parova dvaju tipova
  - `pair<string, string> student("Alan", "Ford");`
  - `pair<int, int> rezultat(3, 2);`
  - `pair<double, double> koordinata(1.1, 3.3);`
- Tipovi ne moraju biti isti
  - `pair<string, int> student("Alan", 5); // ocjena`
  - `pair<string, vector<int> > ime_vektora_i_vektor;`
    - Obratite pažnju na razmak između "> >"
- Potrebno je uključiti zaglavlje `<utility>`
  - `#include <utility>`
- `pair` je predložak (isto kao i `vector`)

pair

- Pristup prvom elementu u `pair`-u
  - `pair<string, int> student("Alan", 5); // ocjena`
  - `cout << student.first;`
- Pristup drugom elementu u `pair`-u
  - `student.second = 3;`
- Možemo kombinirati `pair` i `typedef`
  - `typedef pair<string, int> OcjenaStudenta;`
  - `OcjenaStudenta o; // predstavlja pair ...`

pair

- Zadatak: Napišite funkciju `min()` koja za dobiveni vector `int`-ova vraća najmanji element tog vectora, te broj njegovog pojavljivanja.

```
typedef pair<int,int> min_val_pair;
min_val_pair min(const vector<int>& ivec) {
    int minVal = ivec[0];
    int occurs = 0;
    int size = ivec.size();

    for (int i = 0; i < size; ++i) {
        if (minVal == ivec[i])
            ++occurs;
        else
            if (minVal > ivec[i]) {
                minVal = ivec[i];
                occurs = 1;
            }
    }
    return make_pair(minVal, occurs);
}
```

pair

- Zadatak: Učitavajte znak po znak sa tipkovnice sve dok ne učitate točku, upitnik ili uskličnik. Ako ste pročitali slovo, spremite ga u vektor. Vektor neka se sastoji od uređenih parova <slovo, broj pojavljivanja slova>. Dakle, kada pročitate novo slovo, potrebno je pregledati da li u vektoru već postoji navedeno slovo. Ako postoji, samo treba povećati odgovarajući brojač; inače treba ubaciti novo slovo (sa odgovarajućim brojačem) na kraj vektora.

pair

- Operacije na `pair`-u:
  - `pair<T1, T2> p;`
  - `pair<T1, T2> p(v1, v2);`
  - `make_pair(v1, v2); //stvara i vraća novi par`
  - `p1 < p2 //leksikografsko uspoređivanje`
  - `p1 == p2 //uspoređivanje po koordinatama`
  - `p.first //prva koordinata`
  - `p.second`
- Par dozvoljava direktan pristup svojim koordinatama (za razliku od većine ostalih bibliotečnih tipova)

pair

- Zadatak: Napišite program koji čita niz riječi sa ulaza. Pohranite u vektor parove (riječ, redni broj riječi).
- Zadatak: Napišite program koji učitava matricu, te vraća koordinate maksimalnog elementa matrice (par(int,int)).

Asocijativni spremnik

- Asocijativni spremnici podržavaju gotovo sve operacije kao i slijedni spremnici:
  - Konstruktori:
    - `C<T> c; C<T> c1(c2); C<T> c(b,e);`
  - Relacijski operatori:
    - `== !=`
  - Konstruktori:
    - `c.begin(); c.end();`
  - Tipovi: `value_type`
  - Zamjena: `c1.swap(c2);`
  - Brisanje: `c.clear(); c.erase(i);`
  - Veličina: `c1.size;`

Asocijativni spremnik

- Nepodržane operacije su:
  - `front, push_front, pop_front`
  - `back, push_back, pop_back`
- Elementi su u asocijativnim spremnicima poredani po ključu, bez obzira na redoslijed kojim ih ubacujemo u spremnik

Mapa

- Zaglavlje
  - `#include <map>`
- Mapa je asocijativno polje
  - sadrži parove (ključ, vrijednost)
  - Ključ se upotrebljava za indeksiranje elemenata
  - vrijednost predstavlja korisni podatak koji želimo čuvati
- Primjeri:
  - `map<string, short> ocjene;`
  - `map<string, short> zaposlenik;`
  - `map< pair<int,int>, boja> slika;`

Mapa

- Konstruktori:
  - `map<k, v> m; //mapa (ključ, vrijednost)`
  - `map<k, v> m1(m2); //mapa m1 kao kopija m2`
  - `map<k, v> m(b, e); //mapa preko iteratora`
- Tip koji koristimo kao ključ mora podržavati `operator<`
- Tipovi:
  - `map<k,v>:: key_type` ključ
  - `map<k,v>:: mapped_type` vrijednost
  - `map<k,v>:: value_type` par (const key_type, mapped_type)
  - `map<k,v>::iterator`

Mapa

- Dereferenciranje iteratora na mapu vraća par
- Dodavanje elemenata u mapu:
  - `operator[]`
    - `map<string, int> word_count;`
      `word_count[“abc”] = 1;`
- Korištenje vrijednosti koju vraća operator[]
  - `cout << word_count[“abc”];`
  - `++word_count[“abc”];`
- Za razliku od vektora, operator[] kod mape vraća par(const key_type, mapped_type), a ne samo “mapped_type”
- Oprez: operator[] ubacuje element u mapu ako on tamo još nije bio

Mapa

- Algoritam ubacivanja operatorom[]:
  - Provjeri se da li u mapi već postoji element sa zadanim ključem
  - Ako ne postoji
    - u mapu se ubaci novi par (ključ, vrijednost), ali tako da je vrijednost defaultna (0 u gornjem primjeru)
    - Iz mape se dohvaća par sa traženim ključem, i vrijednost se postavlja na traženu vrijednost (1 u prehodnom primjeru)
  - Ako postoji
    - Iz mape se dohvaća par sa traženim ključem, i vrijednost se postavlja na traženu vrijednost (1 u prethodnom primjeru)

Mapa

- Mapa stvara binarno stablo traženja
  - Lijeva djeca su manja od korijena, a desna su veća
- Stablo je dobro balansirano (red-black tree)
  - Razlike u visini nisu velike
  - Ne može se dogoditi da ubacivanjem sortiranom niza dobijemo stablo koje izgleda kao lista

Mapa

- Zadatak: Napišite program koji učitava riječi sa ulaza te računa i ispisuje broj pojavljivanja pojedinih riječi.

Mapa

- Dodavanje elemenata u mapu:
  - `insert`
    - `m.insert(e)`
      - Ubacuje par e u mapu m ako ključ od e još nije u mapi
      - Ako je ključ od e u mapi, onda se ne dogodi ništa
      - Vraća par (iterator, bool)
        - Iterator pokazuje na ubačeni element
        - Bool kazuje da li je element bio ubačen ili ne
    - `m.insert(b, e)`
      - svaki element u rasponu iteratora b i e se ubacuje u mapu m
      - Vraća void

Mapa

- Primjeri:
  - `word_count.insert(make_pair("abc", 1));`
  - Insert očekuje par
- Dohvat elementa iz mape
  - `int occurs = word_count["xyz"];`
- Problem: ako ključ "xyz" nije postojao u mapi, operator[] ga je upravo dodao u mapu, što ne bismo htjeli

Mapa

- Operacije koje provjeravaju da li je ključ u mapi bez da ga ubace:
  - `m.count(k)`
    - Vraća broj pojavljivanja ključa k u mapi m
    - `Vraća ustvari 0 ili 1`
  - `m.find(k)`
    - Vraća iterator na ključ k, ako k postoji u mapi
    - Ako ključ ne postoji u mapi, vraća end() iterator
- Zadatak: Ubacujte riječi sa ulaza u mapu, ali samo ako riječ već nije u mapi (dobit ćemo ustvari skup riječi na ulazu). Ako riječ već postoji u mapi, ispišite poruku o tome.
  - Ovo se može riješiti i bez count i find

Mapa

- Brisanje elemenata iz mape:
  - `m.erase(k)`
    - Briše elemente sa ključem k u mapi m
  - `m.erase(i)`
    - Briše element na kojeg pokazuje iterator i

- Zadatak: Upišite nekoliko riječi u mapu, te izbrišite sve riječi koje se pojavljuju više od jednom.
  - Pazite: erase može pokvariti iteratore

Mapa

- Zadatak: Ispišite znak i riječ koji su se u nekoj C++ datoteci najčešće pojavljivali.

Skup

- Zaglavlje
  - `#include <set>`
- Skup je samo kolekcija ključeva
- Primjeri:
  - `set<string> rijeci;`
  - `set< pair<int,int> > koordinate;`
- Operacije definirane na skupu su identične operacijama na mapi, osim:
  - Nema mapped_type
  - Nema operatora[]

Skup

- Zadatak: Ispišite skup znakova koji su se pojavili u nekoj C++ datoteci.
- Zadatak: Napišite program koji računa uniju i presjek dva skupa integera.

Multimapa i multiskup

- Zaglavlje
  - `#include <map>`
  - `#include <set>`
- "Multi": dozvoljeno je višestruko pojavljivanje istog ključa
- Primjeri:
  - `multimap<string, int> rijeci;`
  - `multiset<string> tekst;`
- Operacije su iste kao kod mape i skupa, osim:
  - Multimap ne podržava operator[]

Multimapa i multiskup

- Dodatne operacije specifične za “multi”:
  - `m.lower_bound(k);`
  - `m.upper_bound(k);`
  - `m.equal_range();`

Multimapa i multiskup

- Zadatak: Ubacite nekoliko podataka u multimapu koja sadrži parove (autor,djelo). Npr. (“Mato Lovrak”, ”Vlak u snijegu”). Ubacite u multimapu nekoliko djela istog književnika. Isprobajte operacije ...bound i ...range.

Generički algoritmi

- Generički algoritmi su algortimi koji barataju sa spremnikom putem iteratora, ne znajući pritom o kakvom se točno spremniku radi
- Unutar spremnika definirane su samo najvažnije funkcije za rad sa točno određenim spremnikom
- Operacije zajedničke svim spremnicima implementirane su odvojeno u obliku generičkih algoritama
- Zaglavlja
  - `#include <algorithm>`
  - `#include <numeric>`

Generički algoritmi

- Primjeri:
  - Sort, find, merge, fill, count
  - Sort je uvijek isti (dvije petlje, if i swap), bez obzira na tip podataka koji sortiramo
- Najčešći oblici algoritama:
  - `Alg(beg, end, other)`
  - `Alg(beg, end, dest, other)`
  - `Alg(beg, end, beg2, other)`
  - `Alg(beg, end, beg2, end2, other)`

Generički algoritmi

- `accumulate(beg, end, val)`
- `find(beg, end, value)`
  - `find_if()`
- `count()`
  - `count_if`
- `fill`
  - `fill_n`
- `replace(beg, end, what, with)`
  - `replace_if(beg, end, pred, with)`
- `remove(val)`
  - `Remove_if(pred)`
- `sort(beg, end)`
  - `sort(beg, end, pred)`
- Zadatak: Isprobajte gore navedene funkcije

Generički algoritmi

- Operacije specifične za listu
  - Rade brže i bolje nego iste operacije iz <algorithm>
  - `l.remove(val)`
  - `l.remove_if(predikat)`
    - Predikat je funkcija koja vraća vrijednost koju možemo pretvoriti u bool
  - `l.reverse()`
  - `l.sort()`
  - `l.unique()`
  - `l.merge(list2)`
- Zadatak: isprobajte gore navedene funkcije