

**Konteynerlar ro'yxati. Ro'yxat operatsiyalari**

C++ dasturlash tilida dasturning bajarilish vaqtida obyektlarga ishlov berish, ular uchun xotirani belgilash, ularga ehtiyoj bo'lmaganda esa xotirani bo'shatadigan dinamik ma'lumotlar tuzilmalarni yaratish vositasiga ega.

Dinamik ma'lumotlar tuzilmalari - bu kerakli vaqtda xotirani belgilaydigan va bo'shatadigan ma'lumotlar tuzilmasi hisoblanadi.

Dinamik ma’lumotlar tuzilmasining toifalari

Bu yerda hamma STL konteynerli sinflarining umumiy imkoniyatlari ifodalab berilgan. Ko‘pincha vaziyatlarda *talablar* haqida gap yuritiladi, qaysilari hamma STL konteynerlari tomonidan bajarishi shart. Quyida uchta asosiy talab keltirilgan.

- Konteynerlarga element qo‘shilganda, konteyner tashqi ob’ektga murojaatni saqlamay, uning ichki nusxasini yaratadi. Demak, STL konteynerning elementlari nusxalashni qo‘llashi shart. Agar konteynerga saqlanadigan ob’ektning nusxalash ochiq konstruktoriga ega bo‘lmasa yoki ob’ektning nusxalanishi tavsiya etilmasa (masalan: ushbu ob’yekt ko‘p vaqtni sarflasa), bunda konteynerga ko‘rsatgich yoki ushbu ob’yektga murojaat qiladigan ko‘rsatgich ob’yekti kiritiladi.

- Konteyner elementlari aniq tartibda joylashadi. Bu degani, elementlar iterator

orqali qaytib saralanganda, saralash tartibi o‘zgamasligi lozim.

Konteynerning har bir tipida vazifalar aniqlangan, qaysilari elementlarni saralash uchun iteratoratorlarni qaytaradi. Iteratorlar STL algoritmlarning ishlashi uchun asaosiy interfeysi bo‘ladi.

- Umumiy holda konteyner elementlari bilan vazifalar xvfsiz emas. CHaqiradigan tomon vazifaning parametrlari muvofik talablarga javob berishini nazorat qilishi kerak. Qoidalarning buzilishi kutilmagan holatlarga olib keladi. Odatda STL istisnolarni o‘zining kodida *yaratmaydi* *(generatsiya* *qilmaydi).* Ammo, agar istisno foydalanuvchi vazifalari tomonidan yaratilsa, qaysilarini STL konteyneri chaqirsa, bunda vaziyat o‘zgaradi.

Konteynerlar ustida umumiy vazifalar (operatsiyalar)

1.1. jadvalida hamma konteynerlar uchun umumiy operatsiyalar keltirilgan. Umumiy operatsiyalar oldingi qismda ko‘rsatilgan qoidalarga bo‘ysinadi. Bir hillari keyingi qismlarda batafsil yoritilgan.

1.1. jadval**.** Konteynerli sinflarning umumiy operatsiyalari.

Operatsiya

ContType s

~~c(beg,end) element arning nusxalari bilan belgilaydi~~

lekin ba’zan tezroq bajariladi)

Tavsifi

Elementsiz bo‘sh konteynerlarni yaratadi

ContType Konteynerni yaratadi va (beg, end) intervalida uni hamma

s.~ ContType()Hamma~~l~~elementlarni yo‘qotadi va xotiranibo‘shatadi.

c.size() Aslida bo‘lgan elementlar sonini qaytaradi

c.empty() Konteynerning bo‘shligini tekshiradi (size()==0ekvivalenti,

c.max\_size()

cl == c2

Elementlarning maksimal bo‘ladigan sonini qaytaradi

cl va c2 tengligini tekshiradi

cl != c2 cl va c2 tengsizlikni tekshiradi (!(c1==c2) ekvivalent) cl < c2 cl kichik c2, ifodani tekshiradi

cl > c2 cl katta c2 , ifodani tekshiradi (ekvivalent c2<c1)

cl < = c2 cl katta emas c2 (ekvivalent !(c2<c1)), ifodani tekshiradi cl >= c2 cl kichik emas c2 (ekvivalent !(c1<c2)), ifodani tekshiradi

cl = c2 cl ga c2 hamma elementlarni beradi c1.swap(c2) cl va c2 kontentini joylarini almashtiradi

swap(clrc2) c.begin() c.end()

c.rbegin()

iterator qaytar i

~~pozit yaga qayt vchi iter o ni qaytaradi~~

Konteynerdan hamma elementlarni yo‘qotadi (konteyner

Xuddi o‘sha global funksiyaning shaklida Birinchi element uchun iteratorni qaytaradi

Iteratorni oxirgi elementdan keyingi pozitsiyaga qaytaradi

Orqaga qarab saralaganda, birinchi elementi uchun qaytuvchi

c.rend() Orqagani qarab adsaralaganda, oxirgi elementdan keyin

c.insert(pos,elem) elem~~si~~nusxasini~~u~~qo‘yadi~~at~~(~~r~~ qaytarilganqiymatningturli

variantlari bilan va birinchi argumentning interpritatsiyasi

c.erase(beg,end)

c.clear()

c.get\_allocator()

(beg,end) intervalidan hamma elementlarni yo‘qotadi (ba’zi ~~konteynerlar yo‘qotilgan intervaldan keyin keyingi elementni~~

Konteyner xotirasi modelini qaytaradi

**Initsializatsiya**

Turli konteynerli sinf jimlik qoidasiga asosan konstruktorga ega bo‘ladi, qaysiki konstruktorni va destruktorni nusxalaydi. Bundan tashqari, berilgan intervaldan konteynerni elementlar bilan insializatsiyalash imkoniyati mavjud. Muvofiq konstruktor konteynerni boshka konteyner elementlari bilan, massivning yoki shunchaki elementlar bilan initsializatsiyalash imkonini beradi. Bunday konstruktorlar sinfning shablonli funksiyalar shaklida beriladi, shu sababda farqi nafaqat konteyner tipida bo‘lishi mumkin, hamda elementlar tipida bulishi mumkin. Initsializatsiyaning misollari quyida keltirilgan:

Boshqa konteyner elementlari bilan initsializatsiyalash: std::list<int> l; // 1 - int bog‘liqli ro‘yxat

// Ro‘yxatning hamma elementlarini Vectorga nusxalash va float std: :vector<float> sP .begin() J .end()); tipga keltirish

Massiv elementlari bilan initsializatsiyalash:

int array[] = { 2. 3, 17. 33. 45. 77 }: // Massivning hamma elementlarini

std::set<int> c(array,array+sizeof(array)/sizeof(array[0])); to‘plamga nusxalash Standart kirish ma’lumotlarning oqimi yordamida initsializatsiyalash

// standart kirishdan int tipli elementlarni dekga o‘qish

std::deque<int> s((std::istream\_iterator<int>(std::cin)).

(std::istream\_iterator<int>()));

Initsializatorning argumentlarini qo‘shimcha aylana qavsga qo‘yish kerak. Ularsiz ifoda boshqacha ishlaydi, to‘g‘risi quyida ko‘rsatilgan buyruqlarda xatolik paydo bo‘ladi. Usha buyruqni qavslarsiz ko‘rib chiqamiz:

std::deque<int> c(std::istream\_iterator<int>(std::cin). std::istream\_iterator<int>()):

Ushbu variantda c *funksiyani* e’lon qiladi*,* qaysisi deque<int>tipini qaytaradi. Uning cin birinchi parametri tipu istream\_iterator<int> tipiga qarashli, ikkinchi nomsiz parametri esa — “argumentsiz chaqiriladigan funksiya va istream\_iterator<int> ni qaytaradigan” tipga qarashli. Ushbu tuzilma ham e’londay bo‘ladi, ham ifoda deb bilinadi. Demak tilning qoidalari bo‘ycha bu tuzilma e’lon deb qabul qilinadi. Agar qo‘shimcha aylana qavslar kiritilsa, initsializator e’lonning sintaksiga muvofiq bo‘lmaydi.

Argumentlarni boshqa intervaldan o‘zlashtirganda yoki qo‘yganda ham Ushbu usul qo‘llaniladi. Ammo, bunda interfeys ko‘shimcha argumentlari mavjudligi bilan farqlanadi yoki konteynerli sinflarning hammasiyam buni qo‘llab bilmaydi.

**O‘lchamni tekshirish operatsiyalari**

Hamma konteynerli sinflar uchta o‘lchamni tekshirish operatsiyalarini qo‘llaydi. - size(). Funksiya konteynerda joriy elementlar sonini qaytaradi

- empty(). Konteynerning (size()==0) joriy elementlar nol soninini tekshirish qisqartirilgan shakli. Ammo, empty() funksiyasi yanada samarali amallanishi mumkin, shuning uchun imkoni boricha aynan shuni ishlatish lozim

- max\_size(). Funksiya konteynerning ichida bo‘lishi mumkin bulgan

elementlarning maksimal sonini qaytaradi. Son amallanishiga bog‘liq. Masalan, odatda Vectorning hamma elementlari xotiraning birta blokining ichida saqlanadi, bunisi PCga qo‘shimcha cheklovlar qo‘yishi mumkin. Umumiy vaziyatda max\_size() indeks tipi maksimal qiymati bilan bir hil bo‘ladi.

**Solishtirishlar** **(taqqoslar)**

Solishtirish odatiy operatorlari ==, !=, <, <=, > va >= quyda ko‘rsatilgan uchta qoida orqali aniqlanadi:

- Konteynerlarning ikalasi ham birta tipga qarashli bo‘lishi shart.

- Ikkita konteyner teng bo‘ladi, agar ularning elementlari bir hil bo‘lsa va ketma-ketligi bir hil tartibda bo‘lsa. Elementlarning tengligi ==operatori orqali tekshiriladi.

- “Kichik/katta” konteynerlar orasidagi munosabati leksikografik mezoni bo‘yicha tekshiriladi.

**Swap** **funksiyasini** **o‘zlashtirish**

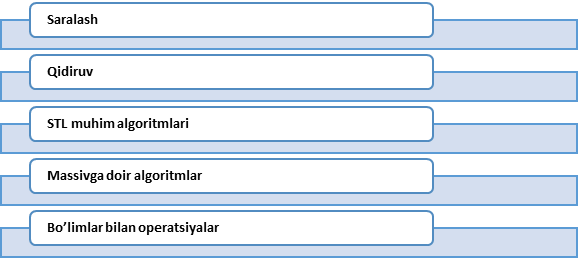
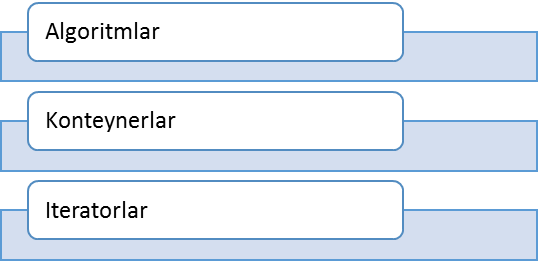
Konteynerlarni o‘zlashtirish jarayonida manba-konteynerning hamma elementlari nusxalanadi, qabul qiluvchi konteynerning elementlari esa hammasi o‘chiriladi. SHunday qilib konteynerlarning o‘zlashtirilishi jqimmatbaho vazifa deb hisoblanadi.

Agar konteynerlar bir tipli bo‘lsa va manba keyinchalik kerak bo‘lmasa, optimallashtirish oddiy usuli bor: swap() funksiyasini amallang.

Swap() funksiyasi juda samarali ishlaydi, chunki konteynerlarning faqat ichki berilganlarni qo‘llaydi. Bundan tashqari, Ushbu funksiya ichki ko‘rsatgichlarni taqdim etadi, qaysilari berilganlarga murojaatlarni o‘z ichiga oladi. Demak, swap() funksiyasi o‘zlashtirishda doimo chiqli emas, bir hil murakkablik bilan bajariladi.

STL (Standard Template Library)

Standart shablonlar kutubxonasi – C++ shablon sinfining to’plami bo’lib, dasturlashning umuniy struktura va funksiyalarini o’z ichiga oladi. Masalan

ro’yxat, stack, massiv va boshqalar. Bu sinflar, algoritmlar va konteynerlar

iteratorlarining kutubxonasi hisoblanadi.

C++ stl 3 ta komponentga ega:

Algoritm nima?

Algoritm sarlavhasi ma’lum oraliq elementlariga qo'llash uchun mo'ljallangan maxsus funksiyalar to'plamiga ega. Bu funksiyalar konteynerga ishlatiladi va konteyner tarkibi uchun turli xil operatsiyalarni bajarishi vositalarini taqdim etadi.

Algoritm nima qila oladi?

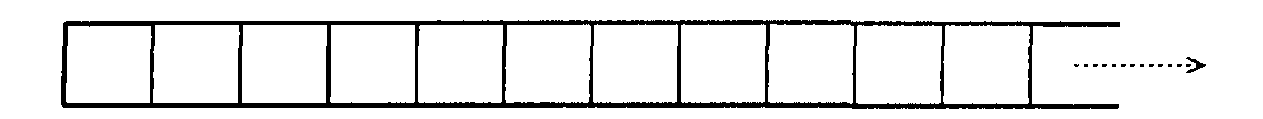
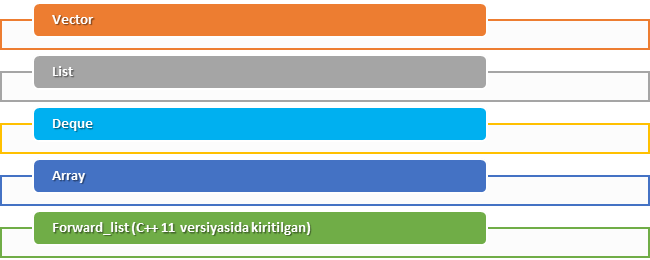
KONTEYNERLAR NIMA?

Konteynerlar yoki sinf konteynerlari ma’lumot va obyektlarni saqlaydi.Quyidagi

konteynerlar mavjud:

Assosiativ konteynerlar: tez topish mumkin bo’lgan, tartiblangan ma’lumotlar

tuzilmasini realizatsiya qilish imkoniyatini beradi.



ITERATORLAR

Ketma-ketlikni qiymatini qayta ishlash uchun foydalaniladi. Interatorlar kolleksiya obyektlarining elementlari orqali o’tish uchun ishlatiladi. Bu kolleksiyalar konteynerlar yoki to’plam osti konteynerlar bo’lishi mumkin.

**1.2.** **KETMA-KET** **KONTEYNERLAR** Ketma-ket konteneylar: ketma-ket murojaat qilinuvchi ma’lumotlar

strukturasini realizatsiya qiladi.

**Vectorlar**

Vector bu abstrakt model, qaysisi elementlar bilan ishlaganda dinamik massivni imitatsiya (o‘xshatish) qiladi. (rasm. 1.2.1). Ammo, standarta Vectorni amallaganda aynan dinamik massiv qo‘llansin deb aytilmagan. To‘g‘riroq bunisi amallarning qiynchiligiga qo‘yilgan talablarga va cheklovlarga asoslanib tanlangan usul.

1.2.1-rasm. Vector ko’rinishi.

Vectorni dasturda ishlatish uchun, unga <vector>sarlavha faylini qo‘shish shart

#inc1ude <vector>

Vectorning tipi std: nomlar doirasida sinf shabloni deb aniqlanadi.

namespace std {

template <class T.

class Allocator = allocator<T> > class vector: }

Vector elementlari ixtiyoriy T tipiga qarashli, bu tip o‘zlashtirishni va nushalashni qo‘llaydi. SHablonning ikkinchi shart bo‘lmagan parametri xotira modelini aniqlaydi. Jimlik qoidasiga asosan allocator modeli ishlatiladi, Ushbu model C++ standart kutubxonasida berilgan.

**Vectorlar** **imkoniyatlari**

Vector elementlari ichki dinamik massivga nushalanadi. Elementlar doimo aniq tartibda saqlanadi, demak Vector *tartibli* *kolleksiya* kategoriyasiga tegishli. Vector o‘z elementlariga *ixtiyoriy* *ruxsatni* ta’minlaydi. Bu degani, anik pozitsiyali turli elementga murojaat doimiy vaqt bilan to‘g‘ridan to‘g‘ri bajariladi. Vectorlar iteratorlari ixtiyoriy ruxsat iteratorlari deb hisoblanadi, bunisi Vectorlarga Hamma STL algoritmlarini qo‘llash imkonini beradi.

Vectorning oxirida elementlarni qo‘shish va o’chirish operatsiyalari juda tz bajariladi. Agar elementlar boshida yoki o‘rtada ko‘shilsa yoki yo‘qotilsa, tezlik pasayadi, chunki keyingi elementlarni yangi joyga ko‘chirish to‘g‘ri keladi. Aslida, har bir keyingi elementi uchun o‘zlashtirish operatori chaqiriladi.

Vectorlarning tezligini ko‘tarish uchun, xotirada Vectorning Hamma elementlarini saqlashga kerak bo‘lgan hajmidan katta hajm ajratiladi. Vectorlarni to‘g‘ri va samarali qo‘llash uchun, Vector o‘lchami va Vector hajmining bog‘likligini tushunish kerak.

Vectorlar o‘lchamni tekshiruvchi standart operatsiyalarni qo‘llaydi size(), empty() i max\_size() . Bularga capacity() funksiyasi qo‘shiladi, bu funksiya ajratilgan joriy xotirada saqlab bo‘ladigan elementlarning maksimal sonini qaytaradi. Agar elementlar soni capadty() funksiya orqali qaytaradigan qiymatidan katta bo‘lsa, bunda Vector ichki xotirani qayta taqsimlaydi.

Vector hajmini inobatga olinishi ikkita sababi bor:

- xotira qayta taqsimlangandan keyin Hamma murojaatlar, ko‘satgichlar va

Vector elementlari iteratorlari bekor qilinadi;

- xotira qayta taqsimlanganda vaqt ketadi.

Demak, agar dastur vetorlar uchun ko‘rsatgichlarni, murojaatlarni yoki iteratorlarni qo‘llasa, bunda Vector hajmi muhim bo‘ladi.

Xotirani qayta taqsimlashdan ushlab qolish uchun, oldindan qayndaydir hajmni rezervatsiya qilib olish lozim, bunisi reserve() funksiya orqali bajariladi. Ushbu funksiya zaxiralangan hajm tugaguncha murojaatlarni bekor bo‘lishdan saqlaydi:

std::vector<int> v: // Bo‘sh Vectorni yaratish v.reserve(80); //80ta element uchun xotirani zaxiralash

Boshqa variantda Vector kerakli elementlar soni bilan initsializatsiya bo‘ladi., bunda konstruktorga qo‘shimcha elementlar beriladi. Masalan, agar raqamli qiymat berilsa, bunda u Vectorning dastlabki o‘lchami deb hisoblanadi:

std::vecton<T> v(5); // Vector yaratilishi va uning

//beshta qiymatlari bilan initsializatsiyasi

// (Jimlik qoidasiga asosan T tipli konstruktor besh marta chaqiriladi)

Buning uchun elementlarning tipini aniqlaganda, albatta Jimlik qoidasiga asosan konstruktor aniqlanilishi shart. Ammo, shuni bilish kerakki, murrakab tiplar uchun, konstruktor bo‘lsa ham, initsializatsiya vaqtni talab qiladi. Agar elementlar initsializatsiyasi faqat xotirani zahiralash uchun bo‘lsa, bunda reserve() funksiyasini qo‘llash lozim.

Vectorlar hajmi konsepsiyasi satrlar hajmi konsepsiyasiga o‘xshash, lekin bitta jiddiy farqi bor: reserve() funksiyasini Vectorlarning hajmini kichirayish uchun chaqirib bo‘lmaydi, satrlar uchun esa bunisi mumkin. Agar reserve() funksiyasining argumenti Vectorning joriy hajmidan kichik bo‘lsa, bunda funksiya e’tiborga olinmaydi. Bundan tashqari, amallash hajmni katta ortirish bilan ko‘paytiradi. Ichki fragmentatsiyani oldini olish uchun, ko‘pgina amallashlar birinchi qo‘yishda xotiraning to‘liq blokini ajratadi (2 Kbaytga yaqin), agar oldin reserve() funksiyasi chaqirilmagan bo‘lsa. Agar dastur kam sonli elementlari

Vectorlar to‘plami bilan ishlasa, bunday strategiya hajmning unumsiz sarflanishiga

olib keladi.

Vectorning hajmi xech qachon to‘g‘riga-to‘g‘ri kamaymaydi, shuning uchun, murojaatlar, ko‘rsatgichlar va iteratorlar elementlar yo‘qotilsada ham rost qoladi (faqat shart bilan: ular modifikatsiyalangan elementlarning oldidan joylashgan pozitsiyaga murojaat qiladi). Lekin qo‘yish (vstavka) murojaatlarni, ko‘rsatgichlarni va iteratorlarni bekor qilishi mumkin.

Qolaversa, Vectorning hajmini kamaytirish bavosita usuli ham bor: agar ikkita Vectorning ichidagi eelementlarini swap() funksiyasi orqali almashtirsak, bunda ularning hajmi o‘zgaradi. Quyidagi funksiya elementlarni saqlab, Vectorning hajmini o‘zgartiradi :

template <class T>

void shnnkCapacity(std: :vector<T>& v) {

std::vector<T> tmp(v); // elementlarni yangi Vectorga nusx alash v.swap S tmp); //vetorlarning ichki berilganlarni qayta qo‘yish

}

Vectorning hajmini ushbu funksiyani chaqirmay kichiraytirish mumkin, buyruqni bajarib:

// v Vectorning T tipi uchun hajmini kichiraytirish std::Vector<T>(v).Swap(v);

Lekin esdan chiqarish kerak emas, swap() funksiyasi chaqirilganda Hamma murojaatlar, ko‘rsatgichlar va iteratorlar yangi konteynerga qaratiladi. Ular dastlab qaysi elementlarga murojaat qilgan bo‘lsa, o‘shalarga murojaat qilishni davom etadi. Demak, shrinkCapacity() funksiyasi chaqirilsa, Hamma murojaatlar, ko‘rsatgichlar va iteratorlar bekor bo‘ladi.

Vectorlar ustida vazifalar (operatsiyalar). Yaratish, nusxalash va o’chirish vazifalari

1.2.2 jadvalda Vectorlarning konstruktor va destruktorlari ko‘rsatilgan. Vectorlar yaratilganda elementlar initsializatsiya bo‘ladi yoki bo‘lmaydi. Agar, faqatgina

o‘lchami uzatilsa, elementlar Jimlik qoidasiga asosan konstruktor orqali yaratiladi.

E’tibor bering: suktlikda konstruktorni aniq chaqirilishi asosiy tiplarni initsializatsiyasini (chunonchi, int) 0 (nol) qiladi.

**1.2.2.** **jadvali.**

Operatsiya

elementlari nusxalanadi)

Vectorni yaratadi

~~c(n,elem) bilan initsializatsiya bo‘la~~

Vector konstruktorlar va destruktorlari

Tavsifi

vector<Elem> s vector<Elem> cl(c2)

vector<Elem> s(n)

Elementsiz, bo‘sh Vectorni

Boshqa Vectorning o‘sha tipli nusxasini yaratadi ( Hamma

Jimlik qoidasiga asosan konstruktor orqali, n elementli

vector<Elem> Vectorni yaratadi, qaysisi elem elementining n nusxalari

vector<Elem>c(beg,Vectorni yaratadi, qaysisi~~di~~(beg,end)intervalining

end) elementlari bilan initsializatsiya bo‘ladi c.~vector<Elem>() Hamma elementlarni yo‘qotadi va xotirani bo‘shatadi

Vectorlarlarni o‘zgartirmaydigan vazifalar

1.2.3- jadvalda vectorlarni o‘zgartirmaydigan vazifalar ko‘rsatilgan.

1.2.3-jadval. Vectorlarni o‘zgartirmaydigan vazifalar.

lekin ba’zan tezroq bajariladi)

sonini qaytarad

hajmini kattalashtiradi

Operatsiyaya c.size()

c.empty()

c.max\_size() Capacity()

Reserve()

cl == c2 cl! = c2 cl < c2 cl > c2 cl <= c2

cl >= c2

Tavsifi

Asl bo‘lgan elementlar sonini qaytaradi Konteynerning bo‘shligini tekshiradi (size()==0 ekvivalenti,

Makismal bo‘ladigan elementlar sonini qaytaradi Xotirani qayta taqsimlamay, makismal bo‘ladigan elementlar

Agar joriy Vector hajmi berilgandanikichik bo‘lsa, Vector

cl va c2 tengligini tekshiradi

cl va c2 (ekvivalent !(c1==c2)) tengsizlikni tekshiradi cl nig c2 dan kichikligini tekshiradi

cl ni c2 dan (ekvivalent c2<c1) kattaligini tekshiradi

cl ni c2 dan (ekvivalent !(c2<c1)) katta emasligini tekshiradi cl ni c2 dan (ekvivalent !(c1<c2)) kichikligini tekshiradi

**O‘zlashtirish**

1.2.4-jadvalda yangi elementlarni o‘zlao‘tirish eskilarini o’chirishi bilan vazifalari

ko‘rsatilgan. assign() funksiyasining to‘plami sinf konstruktorlari to‘plami bilan

muvofiq. O‘zlashtirilganda turli manbalar qo‘llaniladi ( konteynerlar, massivlar,

standart berilganlar kirish oqimi) – xuddi konstruktorlar chaqirilganda qo‘llaniladigan manbalar.

1.2.4-jadval. Vectorlar uchun o‘zlashtirish vazifalari.

Hamma o‘zlashtirish vazifalari konstruktorni sukutlitkda chaqiradi, qaysisi

|  |  |
| --- | --- |
| Operatsiya | Tavsifi |
| cl=c2 | clgac2ningHammaelementlarinio‘zlashtiradi |
| c.assign(n,elem) | Berilganelementning n nusxasinio‘zlashtiradi |
| c.assign(beg,end) | (beg,end) intervaliningelementlarinio‘zlashtiradi |
| cl.swap(c2) | clvac2larningichidaginarsalarinijoylaribilanalmashtiradi |
| swap(cl,c2) | Xuddio‘shanarsa,faqatglobalfunksiyasiuchun |

konstruktorni, o‘zlashtirish operatorini va/yoki element tipi destruktoni nusxalaydi.

Misol:

std::list<Elem> 1: std::vector<Elem> coll:

// coll ga 1 coll.assign (1.begin().1 .end()): // kontenti nusxasini kiritish

**Elementlarga murojaat**

1.2.5-jadvalda vector elementlariga to‘g‘ridan-to‘g‘ri murojaat vazifalari ko‘rsatilgan. C va C++ qabul qilinganday, Vectorning birinchi elementiga 0 to‘g‘ri keladi, oxirgisiga esa – size() - l indeksi. SHunday qilib n elementiga n-1 indeksi muvofiq bo‘ladi. Konstantasiz Vectorlarga Ushbu operatsiyalar elementga murojaatni qaytaradi vashu sababda elementlarni modifikatsiyalashi uchun qo‘llanilishi mumkin (boshqa sabablarga qarab modifikatsiya taqiqlangan bo‘lmagan bo‘lsa sharti bilan).

1.2.5-jadval. Vectorlar elementlariga murojaat

|  |  |
| --- | --- |
| Operatsiya | Tavsifi |
| c.at(idx) | idx indeklsi elementni qaytaradi (indeksning taqiqlangan  qiymati bo‘lsa, out\_of\_range istisno generatsiya bo‘ladi ) |
| s(idx) | Idx indekli elementni qaytaradi (interval tekshirilmaydi!) |
| c.front() | Birinchi elementni qaytaradi (mavjudligi tekshirilmaydi!) |

|  |  |
| --- | --- |
| c.back() | Oxirgi elementni qaytaradi (mavjudligi tekshirilmaydi!) |

Chaqiruvchi tomoni uchun eng asosiysi – elementga murojaat bo‘lganda intervalning tekshirilishi bo‘lish yoki bo‘lmasligi. Bunday tekshiruv faqat at() funksiya orqali amalga oshadi. Agar indeks ruxsat bo‘lgan qiymatlar intervaliga kirmasa, bunda out„of\_range istisnosi generatsiya bo‘ladi. Qolgan funksiyalar tekshiruvsiz bajariladi va intervalli xatoliklar kutilmagan xodisalarga olib keladi. Bo‘sh konteyneri uchun [] operatorning chaqirilishi, front() va back() funksiyalarning chaqirilishi Hamma payt kutilmagan xodisalarga olib keladi.

std; :vector<elem> coll;

coll[5] = elem:

// bo’sh vector!

// bajarilish payti xatosi

std::cout << coll.front (): // bajarilish payti xatosi

Demak, [] operatorini chaqirishdan oldin, indeks ruxsat berilgan qiymatiga ega bo‘lganligini tekshirish lozim. front() yoki back() funksiyalari chaqirilganida esa, konteyner bo‘sh bo‘lmasligi kerak. Misol:

std::vector<Elem> coll: //Bo‘sh Vector! If (coll,size() > 5) { coll[5] = elem: //OK

}

if (I coll.empty()) { coiit « col 1. front(); //OK }

coll.at(5) = elem; // out\_of\_range istisnoni generatsiya qiladi **Iteratorlarni** **olish** **funksiyalari**

Iteratorlarni olish uchun Vectorlar standart to‘plamni qo‘llaydi (1.2.6 jadvali). Vectorli iteratorlar ixtiyoriy ruxsat iteratorlar kategoriyasiga qarashli. Bu degani, Vectorlar bilan ishlaganda Hamma STL algoritmlar qo‘llanilishi mumkin.

1.2.6-jadval. Iteratorlarniolishoperatsiyalari

|  |  |
| --- | --- |
| **Operatsiya** | **Tavsifi** |
| c.begin() | Birinchielement uchunyxtiyoriyruxsatliiteratorniqaytaradi |
| c.end() | Oxirgielementdankeyinpozitsiyasiuchunyxtiyoriyruxsatliiteratorni |
| c.rbegin() | Teskarisaralagandabirinchielementuchunteskariiteratorniqaytaradi |

Aslida iteratorlarning turi amallashda aniqlanadi, ammo Vectorlar uchun iteratorlar oddiy ko‘rsatgichlar dek shakllanadi. Oddiy ko‘rsatgich ixtiyoriy ruxsatli iteratorga qo‘yilgan talablarga muvofiq. Madomiki Vectorning ichki tuzilmasi odatda bu massiv, oddiy ko‘rsatgichlar kerakli amallarni ta’minlaydi. Ammo, iterator ko‘rsatgich bo‘ladi deb hisoblab bo‘lmaydi. Masalan, STL xavfsiz versiyasida, qaysisida interval xatolari va boshqa potensial muammolari tekshiriladi, iteratorlar yordamchi sinf shaklida yaratiladi.

|  |  |
| --- | --- |
| c.rend() | Teskarisaralaganda oxirgielementdankeyinpozitsiyasiuchunteskari  iteratorniqaytaradi |

Iteratorlar shu paytgacha haqki, qachon kichik indeksli element qo‘yiladi yoki yo‘qotiladi va xotira hajmi o‘zgarib, qayta taqsimlansa.

**Elementlarni** **qo‘yish** **va** **o’chirish**

1.2.7-jadvalda vectorlar qo‘llaydigan qo‘yish va o’chirish operatsiyalari ko‘rsatilgan. Odatda, STL qo‘llanganda, argumentlarning to‘g‘riligi chakirish tomonidan ta’minlanadi. Iteratorlar to‘g‘ri pozitsiyaga murojaat qilishi shart, intervalning oxiri oldindan kelishi mumkin emas,bo‘sh konteynerdan elementlar yo‘qotilmasligi kerak.

1.2.7-jadval. Vectorlarni qo‘yishvao’chirishoperatsiyalari

~~yangi elementning ozitsiyasini qaytaradi~~

elem nusxasini Vectorning oxirisiga qo‘shadi

|  |  |
| --- | --- |
| Operatsiya | Tavsif |
| c.insert(pos,elem) | Pos iteratorning pozitsiyasiga elem elementning nushasi qo‘yadi va  Positeratorning pozitsiyasiga~~p~~ n nushalarini elem elementning  nushasi qo‘yadi (va qiymatni qaytarmaydi) |
| c.insert(pos,n,elem) |
| cinsert(pos,beg,end) | (beg,end) intervalining Hamma elementlarning Pos iteratorning  ~~pozitsiyasiga nushasini qo‘yadi (va qiymatni qaytarmaydi)~~ |
| c.push\_back(elem) |
| c.pop\_back() | oxirgi elementni yo‘qotadi (uni qaytarmaydi) |
| c.erase(pos) | Pos iteratorning pozitsiyasida elementni yo‘qotadi va keyingi  elementning pozitsiyasini qaytaradi  (beg,end) intervalining Hamma elementlarini yo‘qotadi va keyingi  elementning pozitsiyasini qaytaradi |
| c.erase(beg,end) |

Samarasini ko‘rib chiqqanda, eslash kerakki, qo‘yish va o’chirish tezroq bjariladi, agar:

~~asosan yaryat ladi)~~

|  |  |
| --- | --- |
| c.resfze(num) | Konteynerni num o‘lchamiga olib keladi (agar bunda size()  kattalashsa, yangi elementlar o‘z konstruktori bilan Jimlik qoidasiga  Konteynerninum o‘lchamiga olib~~i~~keladi(agarbunda size()  kattalashsa, yangi elementlar elem nusxalari deb yaryatiladi) |
| c.resize(num,elem) |
| c.clear() | Hamma elementlarni yo‘qotadi (konteyner bo‘sh qoladi) |

- elementlar qo‘yilsa va o’chirilsa interval oxirida;

- operatsiya qayta tiklanishsiz bajarilishi uchun, chaqiruv paytida konteyner hajmi yetarlicha katta bo’lishi lozim;

- bir nechta ketma-ket chaqiruvlar bilan ishlov berilmay, elementlar majmuiga birta chaqiruv bilan ishlov beriladi.

Elementlarning qo‘shilishi va yo‘qotilishi haq bo‘lmagan murojaat, ko‘rsatgich va iteratorlarning, qaysilari qo‘yish pozitsiyasidan keyin keladigan elementlarga murojaat qiladi. Agar qo‘yish xotirani qayta taqsimlashga olib kelsa, bunda hamma murojaatlar , ko‘satgichlar va iteratorlar haqsiz bo‘ladi.

Vectorlar biror qiymatga ega bo‘lgan elementlarni to‘gridan-to‘gri yo‘qotib bilmaydi. Buning uchun algoritm qo‘llaniladi. Masalan, quyidagi buyruq val qiymatli elementlarni yo‘qotadi:

std::vector<Elem> coll:

// val qiymatli Hamma elementlarni o’chirish coll.erase ( remove (col1.begi n(),col 1.end(). val).

coll,end()):

Keyngi parcha vectordan faqat qiymatli birinchi elementni yo‘qotadi: std::vector<Elem> coll:

// val qiymatli birinchi elementni o’chirish std::vector<Elem>::iterator pos;

pos = find(coll .beginO .coll .end().

val);

if (pos != coll .end()) { col 1.erase(pos);

}

**Vectorlar oddiy massiv sifatida foydalanish**

C++ ning standart kutubxonasining xususiyatlarida Vector elementlari uzluksiz xotira blokida saqlanish kerak deb aytilmagan. Lekin, shu narsa nazarda tutilib, xususiyatlarga kerakli o‘zgarishlar kiritiladi deb hisoblanayapti. Demak, turln i indeksi uchun v Vectorida quyidagi shart oldindan rost:

&v[i] == &v[0] + i

Ushbu shart kafolati bilan bajarilsa, bundan quyidagisi chiqadi: qachonki dasturda dinamik massiv ishlatilsa, vector hamma vaziyatlarda qo‘llanilishi mumkin. Masalan, vectorda oddiy S satrlarning char\* yoki const char\* tipli berilganlarini saqlash mumkin.

std::vector<char> v: // char tipli dinamik massiv kabi // Vectorni yaratish

v.resize(41); // 41 belgi uchun xotirani ajratish //(\0 ham kiritilsin)

strcpy(&v[0],"hello, world"): // S satrni Vectorga nusxalash printf(”$s\n”, &v[0]): // S satr ko‘rinishda Vectorning kontentini

//chiqarish

Albatta, vectorni bunday qo‘llaganda ehtiyotkorlik bo‘lishi lozim. (biroq, dinamik massivlar bilan ishlaganda ehtiyotkorlik doim bo‘lishi kerak). Masalan, vectorning o‘lchami hamma nusxalanayotgan berilganlarni sig‘dirishi kerak, shuni nazorat qilishingiz kerak. Agarda Vectorning kontenti S satrday talab qilinsa, bunda kontent \0 elementi bilan tugatilishi lozim. Ammo, oldingi misolda ko‘rsatilganki dasturda T tipli massiv zarur bo‘lsa, bunda vector[T] qo‘llanilishi mumkin va birinchi elementning adresini uzatishimiz mumkin.

E’tibor bering: birinchi elementning adresi o‘rniga iteratorni uzatish noto‘g‘ri bo‘lar edi. Vectorli iteratorning tipi amallash bilan aniqlanadi; balki oddiy

qo‘rsatgich bilan hech qanday o‘xshash joyi yo‘qdir:

printf("$s\n“. V,beginO); // XATO (ishlab biladi, //ammo tashishni buzadi) printf(XsVn". &v[0]); // OK

**Istisnolarga ishlov berish**

Vectorlarda mantiqiy xatolarni tekshirilishi minimumga keltirilgan. Standart bo‘yicha, istisnolarni faqat at() funksiyasi generatsiya qiladi — indeksatsiya operatorning xavfsiz versiyasi. Bundan tashqari, standart talablari bo‘yicha, faqatgina xotira etishmasa bad\_alloc kabi standart istisnolari bo‘lishi shart yoki foydalanuvchi tomonidan bajarilayotgan vazifalarida istisnoslar.

Vector orqali chaqirilgan funksiyalar istisnolarni ishlab chiqsa, bunda C++ ning standart kutubxonasi quyidagini kafolatlaydi.

- Agar push\_back() funksiyasi orqali element qo‘yilganda istisno paydo bo‘lsa, Ushbu funksiya konteynerga o‘zgarishlarni kiritmaydi.

- Agar nusxalash operatsiyalari istisnolarni ishlab chiqmasa (nusxalaydigan konstruktor va o‘zlashtirish operatori), bunda insert() funksiyasi yoki muvaffaqiyatli bajariladi, yoki o‘zgarishlarni kiritmaydi.

- pop\_back() funksiyasi istisnolarni ishlab chiqmaydi.

- Agar nusxalash operatsiyalari istisnolarni ishlab chiqmasa (nusxalaydigan konstruktor va o‘zlashtirish operatori), bunda erase() va s!eag() funksiyalari istisnolarni ishlab chiqmaydi.

- Swap() funksiyasi istisnolarni ishlab chiqmaydi.

- Agar foydalanilgan elementlar nusxalash operatsiyalari paytida istisnolarni ishlab chiqmasa (nusxalaydigan konstruktor va o‘zlashtirish operatori), bunda turli operatsiya yoki muvaffaqiyatli bajariladi, yoki o‘zgarishlarni konteynerga kiritmaydi. Bunday elementlar “oddiy berilganlar” bo‘lishi mumkin, bu degani C++ ixtisoslashgan imkoniyatlardan foydalanilmaydigan tiplar. Masalan, oddiy struktura S “oddiy berilganlar” bo‘ladi.

Destruktorlar istisnolarni ishlab chiqarmaydi: Ushbu da’vo Hamma oldinda keltirilgan kafolatlarni tasdiqlaydi.

**Vectorlarni ishlatish misollari**

Quyida vectorlarni qo‘llash oddiy misoli keltirilgan.

// cont/vectorl,cpp #include <iostream> finclude <vector> finclude <string> finclude <algorithm> using namespace std:

int main() {

// vector<string> sentence satrlarni saqlash uchun bo‘sh Vectorni yaratish;

// Xotirani beshta element uchun zaxiralaymiz, qayta taqsimlashni bartaraf qilish uchun.

sentence.reserve(5);

// bir nechta elementlarni qo‘shish sentence.push\_back("Hello."); sentence. push\_back (’’how"); sentence.push\_back("are"); sentence.push\_back("you"); sentence.push\_back("?");

// probel bilan bo‘lingan elementlarni chiqarish copy (sentence.begin(). sentence.end(). Ostream\_terator<string>(cout.” "));

cout << endl;

// « xizmatchi berilganlarni» chiqarish

cout << ” max\_size(): " << sentence.max\_size() << endl; cout << "size(): " << sentence.size()<<endl;

cout << "capacity(): " <<sentence.capacity0 << endl; // ikkinchi va to‘rtinchi elementlarining joyi almashuvi swap (sentence[l]. sentence[3]);

//’’always" elementni "?'’ elementi oldidan qo‘yish

sentence.insert (find(sentence.begin().sentence.end(),"?"). "always"): // Oxirgi elementga ”!" ni o‘zlashtirish

sentence.back () = ”!"

// probel bilan bo‘lingan elementlarni chiqarish

copy (sentence, begin(), sentence.end(). ostreanijterator<string>(cout." ")): cout << endl; // "xizmatchi berilganlarni" qayta chiqarish

cout << " max\_ size (): " << sentence.max\_size() << endl; cout << ” size(): " << sentence.size() << endl;

cout << " capacity(); " << sentence.capacity()<< endl; }

Dastur bajarilishi natijasi deyarli quyidagicha bo‘ladi:

**Hello,** **how** **are** **you?** **max\_size():** **268435455** **size():** **5** **capacity():** **5** **Hello,** **you** **are** **how** **always** **!** **max\_size():** **268435455** **size():** **6** **capacity():** **10**

E’tibor bering, max\_size() va capacity() qiymatlari amallari bilan bog‘liq. Masalan, Bu vaziyatda ko‘rinib turibdiki, agar Vector zaxiralgan xotiraga sig‘masa, amallash uning hajmini ikkiga ko‘paytiradi.

**Vector<bool> sinfi**

C++ standart kutubxonasida mantiqiy elementlarni o‘z ichiga oladigan Vectorlar uchun vector sinfi ixtisoslashtirilgan. Bunisi nima uchun qilingan? Optimallangan versiya, standart vector funksiyasi bool tipi uchun amallashidan kam joy egallasin. Amallash standart versiyasida har bir elementi uchun minimum 1 bayt zaxiralanadi. vector<bool> ixtisoslashgan amallashda ichki ifodalashda har bir element odatda bir bit bilan ifodalanadi., shu sababda bu amal xotirada sakkiz barobar kam joy egallaydi. Lekin, ammalash tekin bo‘lmaydi: C++da minimal manzilli qiymat minimum 1 bayt hajmga ega bo‘lishi shart. Demak, ixtisoslashgan versiyasi murojaatlarni va iteratorlarni maxsus ishlov berishini talab qiladi.

Natijada vector<bool> sinfi boshqa Vectorlar Hamma talablarini qondirmaydi (chunonchi vector<bool>preference qiymati 1-to‘la qimmatga ega emas, vector<bool>:;iterator iteratori esa ixtiyoriy ruxsatli iterator emas) demak, shablon kodi turli tipli Vectorlar bilan ishlaydi., faqat bu erda bool istisno bo‘ladi. YAnada vector<bool> sinfi oddiy realizatsiyalar ishlash vaqtidan past keladi, chunki elementli amallarni bitli amallarga aylantirish kerak bo‘ladi. Lekin,

vector<bool>ning ichki tuzilishi amallashga bog‘liq, shu sababda samarasi turli

~~bitni belgilash)~~

|  |  |
| --- | --- |
| Operatsiya | Tavsifi |
| c.flip() | Hamma mantiqiy elementlarni inversiya qiladi |
| m[idx].flip() | Mantiqiy berilgan indeks bilan elementni inversiya qiladi |
| m[idx] = val | idx indeksli mantiqiy elementga qiymatni belgilaydi (birta  idx2indekli elementga idx1 indeksli elementningqiymatini  o‘zlashtiradi |
| m[idx1] =  m[idx2] |

bo‘lishi mumkin.

Hisobga oling, vector<bool> sinfi faqat vector<> bool uchun ixtisoslashgan sinfi emas. Uning ichiga maxsus operatsiyalarni qo‘llanilishi kiritilgan, qaysilari bayroqlar va bitlar bilan ishni soddalashtiradi.

vector<bool> konteynerning o‘lchami dinamik tarzda o‘zgaradi, shuning uchun uni dinamik o‘lchamli bitli maydon deb ko‘rishimiz mumkin. Boshqacha deb aytganda, siz bitlarni qo‘shib va yo‘qotib bilasiz. Agar siz statik o‘lchamli bitli maydon bilan ishlasangiz, unda vector<bool> sinfi o‘rniga bitset sinfini ishlatish lozim.

vector<bool> konteynerning qo‘shimcha operatsiyalari 6.8 jadvalida ko‘rsatilgan. flip() mantiqiy inversiyani bajaradigan operatsiyasi, Vectorning Hamma bitlariga, hamda alohida bitiga qo‘llanilishi mumkin.

1.2.8-jadval. vector<bool> maxsusoperatsiyalari

Birta mantiqiy element uchun flip() ni chaqirish mumkin. Birinchi qaraganda bu qiziq ko‘rinadi, chunki indeksatsiya operatori bool tipli qiymatni qaytaradi, flip() ni chaqirilishi esa Ushbu asosiy tipi uchun mumkin emas. Bu erda vector<bool > sinfi standart metodikasini qo‘llaydi, qaysisi guyo *o‘rinbosarlarni* ishlatishiga asoslanadi: vector<bool> uchun indeksatsiya operatori uchun qaytariladigan qiymat yordamchi sinfday taxt qilingan. Agarda siz qaytariladigan qiymatni bool tipli deb izohlamoqchi bo‘lsangiz, bunda tipni avtomatik tarzda o‘zgartiring. Boshqa operatsiyalar uchun sinf funksiyalari aniqlanadi.

vector<bool> e’lonning muvofiq qismi quyidagicha bo‘ladi:

namespace std { \*

class vector<bool> { public;

// Indeksatsiya operatori uchun yordamchi tipi class reference {

public;

// tipni boolga avtomatik tarzda o‘zgartirish operator bool() const; // O‘zlashtirish

reference& operator= (const bool); reference& operator= (const references); // Bitning inversiyasi

void flip(): }

..........

*//* elementlarga qaytish operatsiyalari

// - bool o‘rniga reference tipi qaytariladi reference operator[](size\_type n); reference at(size\_type n);

reference front(); reference back();

} }

Ushbu qismda ko‘rinib turibtiki, elementlarga murojaat qilgan funksiyalar reference tipini qaytaradi. Demak, siz quyidagi buyruqlarni qo‘llashingiz mumkin:

s.front().flip(); // Birinchi mantiqiy elementining inversiyasi c.at(5) = s.back(); // 5 indekli elementga oxirgi elementni o‘zlashtirish

Odatda, chaqirish tomoni Vectorning birinchi, oxirgi va oltinchi elementlarining bo‘lishini nazorat qilishi shart.

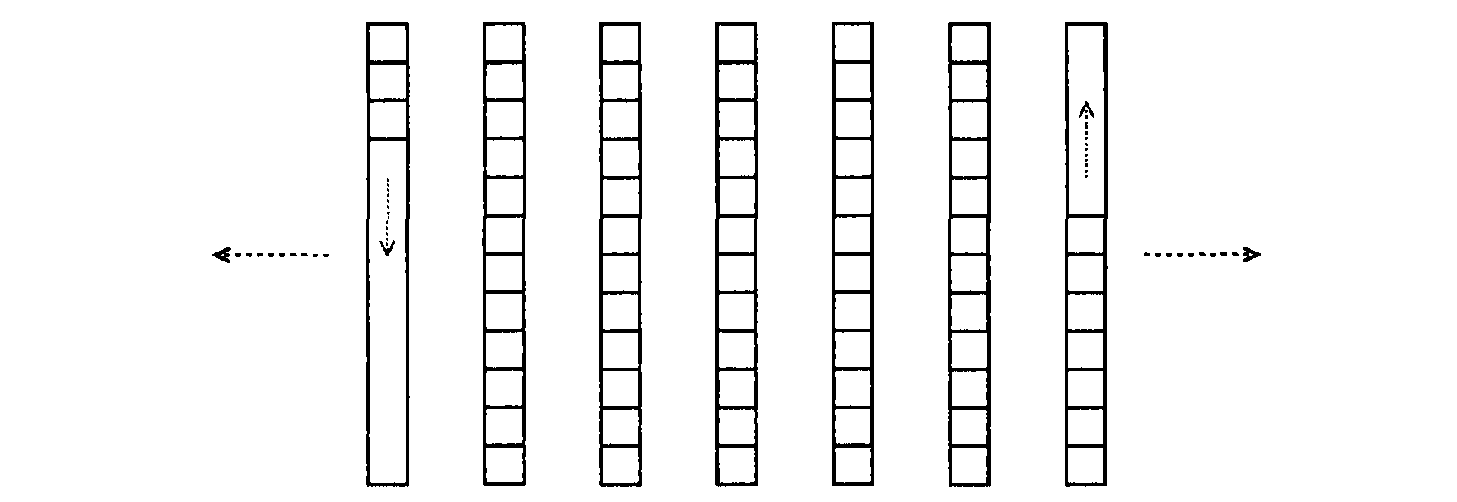
reference ichki tipi faqatgina nokonstanatali vector<bool> tipli konteynerlari uchun qo‘llaniladi. Konstantali elementlarga murojaat funksiyalar oddiy bool tipli qiymatlarni qaytaradi.

**Deklar**

Dek juda ham vektorga o‘xshashli. Dek ham dinamik massivday shakllangan elementlar bilan ishlaydi, ixtiyoriy ruxsatni qo‘llaydi va amalda xuddi shuday interfeysga ega. Farqi o‘shandaki, dekning dinamik massivi ikkala tarafdan ham ochiq. SHu tufayli dek qo‘yish va o’chirish operatsiyalarni boshidan ham, oxiridan ham tez bajaradi (rasm. 1.2.9).

1.2.9-rasm. Dekning mantiqiy tuzilishi

Odatda dek bloklar to‘plamida amallanadi; birinchi va oxirgi bloklar qarama-qarshi qo‘shiladi (rasm 1.2.10)



1.2.10 rasm.Dekning ichki tuzilmasi.

Dasturda dekni ishlatish uchun, unga <deque> sarlavxa faylini qo‘shish lozim: #include <deque>

Dekning tipi std nomlar doirasida sinf shabloniday aniklanadi: namespace std {

template <class T.

class Allocator = allocator<T> > class deque; }

Vektor kabi dek elementlari tipi shablonning birinchi parametrida uzatiladi. O‘zlashtirish va nusxalashni qo‘llaydigan turli tipiga ruxsat. SHart bo‘lmagan shablonning ikkinchi parametri xotiraning modelini aniqlashtiradi, sukutlikda allocator modeli qo‘llaniladi.

**Deklarning imkoniyatlari**

Deklarning imkoniyatlari vektorlarning imkoniyatlaridan ko‘ra ko‘pgina

ajralib turadi.

- Qo‘yish va o’chirish boshida ham, oxirida ham tez bajariladi (vektorlar uchun bunisi faqat oxirida bo‘ladi). Operatsiyalar amortizatsiyalangan o‘zgarmas vaqti bilan bajariladi.

- Ichki tuzilma murojaatlarning qo‘shimcha qatlamini o‘z ichiga oladi, shuning uchun elementlarga murojaat va iteratorning o‘tishi deklarda sekinroq bo‘ladi.

- Iteratorlar aqlli, alohida turli ko‘rsatgichlari bo‘lishi shart. Oddiy iteratorlar to‘g‘ri kelmaydi, chunki blokdan blokga o’tish kerak.

- Xotira bloklari cheklangan tizimlarida (masalan PC ba’zi tizimlarida) dek ko‘proq elementlarni o‘z ichiga olishi mumkin), chunki u xotiraning bitta bloki bilan cheklanmaydi. Demak, max\_size() funksiyasi bloklar uchun katta qiymatni qaytarishi mumkin.

- Deklar xotirani qayta taqsimlash paytini va hajmini boshqarishga yo‘l qo‘ymaydi. Xususan, qo‘yish va yo’qotish turli operatsiyalar paytida, qaysilarining bajarilishi vektorning boshida ham emas, oxirida ham emas hamma ko‘rsatgichlar, murojaatlar va iteratorlar dek elementlariga murojaat qilganda haqsiz bo‘ladi.

- Ammo, xotiraning qayta taqsimlanishi umumiy holda, vektorlarga qaraganda unumliroq bo‘ladi, shuning uchun deklarning ichki tzulishi uchun ular xamma elementlarni nusxalaydi.

- Ishlatilmaydigan bloklarning bo‘shalishi dek xotiraning hajmini kamayishiga olib keladi (bu holat amalashga bog‘liq bo‘ladi).

Keynigi vektorlarning xususiyatlari deklar uchun ham xos bo‘ladi.

- Konteynerning o‘rtasida elementlarning qo‘yilishi va yo‘qotilishi sekinroq bajariladi, shuning uchun joyni bo‘shatish uchun yoki bo‘shlikni to‘ldirish uchun elementlarni ikki tarafdan ham ko‘chirishga to‘g‘ri keladi.

- Iteratorlar ixtiyoriy ruxsatli iteratorlar bo‘ladi. Xulosa. Agar quyidagi shartlar bajarilsa dekni tanlang:

- Elementlarning qo‘yilishi ikki tarafdan ham bajarilsa;

- Dasturda konteyner elementlariga murojaatlar bo‘lmasa;

- Konteyner ishlatilmaydigan xotirani bo‘shatib berishi muhim.

Vektor va deklar konteyner interfeyslari deyarli farq qilmaydi. Agar dasturda vektorning maxsus xususiyatlari ishlatilmasa, siz ikala konteynerni ham ishlatib ko‘rishingiz mumkin.

**Deklar** **ustida** **operatsiyalar**

1.2.7- 1.2.11-jadvallarda deklar qo‘llaydigan xamma operatsiyalar ko‘rsatilgan. **1.2.11-jadval.** Deklar konstruktor vadestruktorlari

|  |  |
| --- | --- |
| Operatsiya | Tavsifi |
| deque<Elem> s | Elementsiz, bo‘sh dekni yaratadi |
| deque<Elem> d(c2) | Bir hil tipli boshqa dekning nusxasini yaratadi  (xamma elementlarini nusxalaydi) |
| deque<Elem> s(n) | Sukutlikda konstruktor yordamida n-elementli dekni yaratadi |
| deque<Elem> s(n, elem) | Elem elementning n-nusxali initsializatsiya  qilinadigan dekni yaratadi |
| deque<Elem> c(beg,end) | [beg,end) intervalning elementlari bilan  initsializatsiya qilinadigan dekni yaratadi |
| с.≈deque<Elem>() | Hamma elementlarni yo’qotib, xotirani bo‘shatadi |

**1.2.12-jadval.**Deklarustidao‘zgartimaydiganoperatsiyalar.

|  |  |
| --- | --- |
| Operatsiya | Tavsifi |
| c.size() | Aslida bo‘lga elementlar sonini qaytaradi |
| c.empty() | Konteyner bo‘shligini tekshiradi (size()==0 ekvivalenti) |
| c.max\_size() | Maksimal bo‘ladigan elementlar sonini qaytaradi |
| cl == c2 | C1 va c2 tengligini tekshiradi |
| cl != c2 | C1 va c2 teng emasligini tekshiradi (!(c1==c2) ekvivalenti) |
| cl < c2 | C1ning c2dan kichikligini tekshiradi |

|  |  |
| --- | --- |
| cl > c2 | C1ning c2dan kattaligini tekshiradi (c2<c1 ekvivalenti) |
| cl <= c2 | C1ning c2dan katta emasligini tekshiradi (!(c2<c1)  ekvivalenti) |
| cl >= c2 | C1ning c2dan kichik emasligini tekshiradi (!(c1<c2)  ekvivalenti) |
| c.at(idx) | Idx indeksli elementni qaytaradi |
| c[idx] | Idx indeksli elementni qaytaradi (intervalni tekshirmasdan!) |
| c.front() | Birinchi elementni qaytaradi (uning borligini tekshirmasdan!) |
| c.back() | Oxirgi elementni qaytaradi (uning borligini tekshirmasdan!) |
| c.rbegin() | Teskariga saralaganda birinchi elementi uchun teskari  iteratorni qaytaradi |
| c.rend() | Teskariga saralaganda oxirgi elementdan keyingi pozitsiyaga teskari iteratorni qaytaradi |

**1.2.13-jadval**. Deklarustidao‘zgartiradiganoperatsiyalari

|  |  |
| --- | --- |
| Operatsiya | Tavsifi |
| c1 = c2 | cl ning xamma elementlarini c2 xamma elementlarini o‘zlashtiradi |
| c.assign(n,elem) | elem elementining n – nuxalarini o‘zlashtiradi |
| c.assign(beg,end) | [beg,end) intervalning elementlarini o‘zlashtiradi |
| cl,swap(c2) | cl va c2 kontentini almashtiradi |
| swap(d,c2) | Xuddi o‘sha, lekin global funksiya shaklida |
| c.insert(pos,elem) | elem ning nusxasini iteratorning pos pozitsiyasiga qo‘yadi va  yangi elementning pozitsiyasini qaytaradi  Elem elementning n – nuxalarini iteratorning pos pozitsiyasiga  qo‘yadi (qiymatini qaytarmaydi) |
| c.insert(pos,n,e1em  ) |
| s. i nsert( pos,  beg,end) | [beg,end) intervalning xamma elementlarining nusxasini  iteratorning pos pozitsiyasiga qo‘yadi (qiymatini qaytarmaydi) |
| c.push\_back(elem) | Dekning oxiriga elem nusxasini qo‘shadi |
| c.pop\_back() | Oxirgi elementni qaytaradi (uni qaytarmaydi) |
| c.push\_front(elem) | elem ning nusxasini dekning boshiga qo‘shadi |
| c.pop\_front() | Birinchi elementni yo‘qotadi (uni qaytarmaydi) |

Deklar operatsiyalari vektorlar operatsiyalaridan faqat quyidagi nisbatlarda

|  |  |
| --- | --- |
| c.erase(pos) | iteratorning pos pozitsiyasidan elementni yo’qotib, keyingi elementning pozitsiyasini qaytaradi |
|  |  |
| c.resize(num) | Konteynerni num o‘lchamiga keltiradi (agar bunda size()  kattalashsa, sukutlikda o‘z konstruktori yangi elementlar  Konteynerni num o‘lchamiga keltiradi (agar bunda size()  kattalashsa, yangi elementlar elem nusxalari dek yaratiladi) |
| c.resize(num,elem) |
| c.clear() | Hamma elementlarni yo‘qotadi (konteyner bo‘sh qoladi) |

farq qiladi:

- (capacity() va reserve()) xajmi bilan bog‘lik funksiyalarni deklar qo‘llamaydi;

- Deklarda birinchi elementni to‘g‘ridan-to‘g‘ri qo‘yish va o’chirish funksiyalar aniq pervogo elementa (push\_front() i pop\_back()).

Qolgan operatsiyalar bir xil ketadi va bu yerda ko‘rsatilmagan. Deklar bilan ishlaganda quyidagilarni inobatga olish shart:

- Elementlarga murojaat qiladigan funksiyalar indekslarning va iteratorlarning to‘g‘riligini tekshirmaydi (at() dan boshqa).

Elementlarning qo‘yilishi yoki yo‘qotilishi xotirani qayta taqsimlanishiga olib kelishi mumkin. Bu degani, turli qo‘yilishda yoki yo‘qotilishda hamma murojaatlar, ko‘rsatgichlar va iteratorlar dekning boshqa elementlariga murojaat qilganda haqsiz bo‘lishlari mumkin. Ushbu da’vo elementlarni dekning boshiga va oxiriga qo‘yganda to‘g‘ri kelmaydi – bu holatda murojaatlar va ko‘rsatgichlar (iteratorlar emas!) haqli bo‘ladi.

**Istisnolarga** **ishlov** **berish**

Aslida deklarda istisnolarga ishlov berish vektorlarda istisnolarga ishlov berish bilan bir xil, push\_front() va pop\_front() qo‘shimcha funksiyalari push\_back() va ror\_ back() funksiyalariga o‘xshash. Shunday qilib C++ standart kutubxonasi quyidagilarni ta’minlaydi:

- Agar push\_back() yoki push\_front() funksiyasi bilan element qo‘yilganda, istisno paydo bo‘lsa bu funksiyalar o‘zgartirish kiritmaydi;

- pop\_back() va pop\_front() funksiyalari istisnolarni ishlab chiqarmaydi.

**Deklarni** **qo‘llashga** **misollar** Quyida deklarni qo‘llash oddiy misoli keltirilgan:

// cont/dequel.srr #include <iostream> #include <deque>

#include <string>

#include <algorithm>

using namespace std: int main()

{

// satrlarni saqlash uchun bo‘sh dekni yaratish deque<string> coll;

// bir nechta elementlarni qo‘yish coll.assign (3, string ("string"));

col 1.push back ("last string"); coll ,push\_front ("first string");

// Yangi satr belgisi bilan elementlarni chiqarish copy (coll.begin(). coll.end(),

ostream\_ iterator<stri ng>(cout.“\n")); cout << endl; // birinchi va oxirgi elementni o’chirish

coll ,pop\_front(); coll.pop\_back();

// "another" ni birinchi elementdan boshqa, xamma elementlarga qo‘yish for (unsigned i=l; i<coll .size(); ++i) {

col 1[i] = "another " + col 1[i]; }

// o‘lchamni 4 elementgacha ko‘paytirish coll.resize (4, "resized string");

// Yangi satr belgisi bilan elementlarni chiqarish copy (coll .begin(). coll.end(). ostream\_iterator<string>(cout. " \ n"));

cout << endl;

{

Dastur bajarilgandan keyin natija quydagicha bo‘ladi: first string

string string string last string

string

another string another string resized string

**Ro‘yxatlar**

STL kutubxonasining list konteynerning elementlari ikki bog‘lamli ro‘yxatga birlashtirilgan (rasm 1.2.14.). Odatdagiday ushbu amallash C++ standart kutubxonasida ko‘rsatilmagan, lekin bu tipli konteynerga cheklovlar va talablar qo‘yilgan.

**Rasm** **1.2.14.** Ro‘yxat tuzilishi

Ro’yxatni dasturda ishlatish uchun, unga <list> sarlavha faylini qo‘yish shart:

#include <vector>

Ro‘yxatning tipi sinf shabloniday std nomlar doirasida aniqlanadi: namespace std {

template <class T,

class Allocator = allocator<T> > class list; }

Ro’yxat elementlari T ixtiyoriy tipiga qarashli. Ushbu tip o‘zlashtirishni va nusxalashni qo‘llaydi. Shart bo‘lmagan shablonning ikkinchi parametri xotiraning modelini aniqlaydi. Sukutlikda C++ning standart kutubxonasida berilgan allocator

modeli qo‘llaniladi.

**Ro‘yxatlarning** **imkoniyatlari**

Ro‘yxatlarning ichki tuzilmasi vektorlar va iteratorlardan to‘liq farq qiladi. Asosiy farqlari quyida ko‘rsatilgan:

- Elementlarga ixtiyoriy ruxsatni ro‘yxatlar qo‘llamaydi. Masalan, beshinchi elementga murojaat qilish uchun, oldingi to‘rta elementni murojaatlar bo‘yicha saralab chiqish kerak. Bu degani, ro‘yxatning ixtiyoriy elementiga murojaat sekin bajariladi.

- Elementlarning qo‘yilishi va yo’qotilishi turli pozitsiyada tez bajariladi. Elementlarni qo‘yish va o’chirish vaqtning doimiy qiymati bilan bajariladi, chunki bu erda boshqa elementlarning ko‘chirilishi talab qilinmaydi. Ichki amallashda, faqatgina bir nechta ko‘rsatgichlarning qiymatlari o‘zgaradi.

- Elementlarni qo‘yishi va yo‘qotilishi natijasida, boshqa elementlarga qarashli ko‘rsatgichlar, murojaatlar va iteratorlar haqli qoladi.

- Ro‘yxatlarda istisnolarga ishlov berish shunday amallanganki har bir operatsiya muvaffaqiyatli tugaydi yoki o‘zgarishlarni kiritmaydi. Boqacha qilib aytganda, ro‘yxat ikki holatning orasida qolmaydi, qayerda operatsiya yarmigacha bajarilgan bo‘lib turadi.

Ushbu prinsipial farqlar ro‘yxatlar funksiyalarida ko‘rsatilgan, bunisi ularni vektorlar va deklar funksiyalariga o‘xshatmaydi.

- Ro‘yxatlarda indeksatsiya operatori va at() funksiyasi aniqlanmagan, chunki ro‘yxatlar ixtiyoriy ruxsatni qo‘llamaydi.

- Ro‘yxatlar hajm va xotirani qayta taqsimlash operatsiyalar bilan bog‘liq bo‘lgan operatsiyalarni qo‘llamaydi, chunki bunday operatsiyalar kerak emas. Har bir element o‘z xotiraga ega va bu xotira yo‘qotilgancha haq bo‘lib turadi.

- Ro‘yxatlar elementlarni ko‘chirish uchun ko‘pgina maxsus funksiyalarni qo‘llaydi. Sinfning bu funksiyalari nomdosh bo‘lgan universal algoritmlarning versiyalarni ifodalaydi.

Ro‘yxatlar ustida operatsiyalar

Yaratish, nusxalash va o’chirish operatsiyalari

Ro‘yxatlar qolgan xamma ketma-ket konteynerlar kabi yaratish, nusxalash

va o’chirish operatsiyalari to‘plamini qo‘llaydi. 1.2.15 jadvalida ro‘yxatlarning konstruktor va destruktorlari ko‘rsatilgan.

**1.2.15** **jadvali.** Ro‘yxatlarning konstruktor va destruktorlari

Operatsiya Tavsifi

elementlar nusxalanadi)

~~c(n,e!em) ro‘yxatni yaratadi~~

~~c(beg,end) ro‘yxatni yaratadi~~

list<Elem> s Elementsiz, bo‘sh ro‘yxatni yaratadi

list<Elem> d(c2) O‘sha tipli boshqa ro‘yxatni nusxasini yaratadi (xamma

list<Elem> s(n)

list<Elem>

N- elementli ro‘yxatni tuzadi, qaysisi sukutlikda konstruktor

orqali yaratiladi

elem elementininig n-nusxalari bilan initsializatsiya bo‘ladigan

list<Elem>[beg,end) interva elementlaribilan initsializatsiya bo‘ladigan

c.~list<Elem>()Hamma elementlarni yo‘qotibxotirani bo‘shatadi

**Ro‘yxatni** **o’zgartirmaydigan** **operatsiyalar**

Ro‘yxatlarda o‘lchamni aniqlash va taqqoslash uchun standart operatsiyalar to‘plami qo‘llaniladi. Ushbu operatsiyalar 1.2.16 jadvalda ko‘rsatilgan.

**1.2.16** **jadval.**

|  |  |
| --- | --- |
| Operatsiya | Tavsifi |
| c.size() | Elementlarning asl sonini qaytaradi |
| c.empty() | Konteyner bo‘shligini tekshiradi (size()==0 ekvivalenti) |
| c.max\_size() | Maksimal bo‘ladigan elementlar sonini qaytaradi |
| cl == c2 | C1 va c2 tengligini tekshiradi |
| cl! = c2 | C1 va c2 teng emasligini tekshiradi (!(c1==c2) ekvivalenti ) |
| cl < c2 | C1ning c2dan kichikligini tekshiradi |
| cl > c2 | C1ning c2dan kattaligini tekshiradi (c2<c1 ekvivalenti) |
| cl <= c2 | C1ning c2dan katta emasligini tekshiradi (!(c2<c1) |

|  |  |
| --- | --- |
| cl >= c2 | C1ning c2dan kichik emasligini tekshiradi (!(c1<c2) |

**O‘zlashtirish**

Ro‘yxatlar ketma-ket konteynerlari uchun oddiy o‘zlashtirish operatsiyalar ro‘yxatini ham qo‘llaydi ( 1.2.17 jadval).

**1.2.17** **jadval.** Ro‘yxatlaruchun o‘zlashtirish operatsiyalari.

|  |  |
| --- | --- |
| Operatsiya | Tavsif |
| cl == c2 | Cl ga c2 elementlarini o‘zlashtiradi |
| c.asslgn(n,elem) | Elem elementning n-nusxalarini beradi |
| c.assign(beg,end) | [beg,end) intervalning elementlarini beradi |
| d.swap(c2) | cl va c2 kontentini joylarini almashtiradi |
| swap(cl,c2) | Xuddi o‘sha, faqat global funksiya doirasida |

Odatdagiday, o‘zlashtirish operatsiyalari initsializatsiyasi turli manbalardan bo‘lgan konstruktorlarga muvofiq.

**Elementlarga** **murojaat**

Ro‘yxatlar elementlarga ixtiyoriy ruxsatni qo‘llamaydi, shu sababda elementlarga to‘g‘ridan-to‘g‘ri murojaat qilish uchun front() va back() funksiyalari qo‘llaniladi. Ush bu funksiyalar 1.2.18 jadvalida keltirilgan.

**1.2.18** **jadval.** Ro‘xatningelementlarigato‘g‘ridan-to‘g‘rimurojaatoperatsiyalari

|  |  |
| --- | --- |
| **Operatsiya** | **Tavsifi** |
| c.front() | Brinchi elementni qaytaradi (mavjudligini tekshirmasdan!) |
| c.back() | oxirgi elementni qaytaradi (mavjudligini tekshirmasdan!) |

Oldingi vaziyatlardagiday, ushbu operatsiyalar konteynerda elementlar mavjudligini *tekshirmaydi.* Agar konteyner bo‘sh bo‘lsa, ularning chaqirilishi kutilmagan natijaga olib keladi. Demak ularni chaqirishdan oldin, konteyner bo‘sh emasligini tasdiqlash shart. Misol:

std::list<Elem> coll; // Bo‘sh ro‘yxat!

std: :cout << coll.front(); // BAJARISH VAQTI XATOSI

if (coll .empty()) {

std::cout <<coll.back(); // OK

}

**Iteratorlarni** **olish** **funksiyalari**

Ro‘yxatning ixtiyoriy elementiga murojaati faqat iteratorlar orqali bajariladi. Ro‘yxatlar iteratorlarni olish standart funksiyalarni qo‘llaydi (1.2.19 jadval). Unga qaramasdan ixtiyoriy ruxsatning yo‘qligi iteratorlarni faqatgina ikki yo‘nalishli qiladi. Bu degani, ixtiyoriy ruxsat bilan ishlaydigan iteratorlar algoritmini chaqirib bo‘lmaydi. Ushbu kategoriyaga saralash algoritmlari qarashli. Bunga qaramay ro‘yxatlarda elementlarni saralash uchun sort() maxsus funksiya qo‘llaniladi.

**1.2.19** **jadval.** Iteratorlarniolishoperatsiyalari

**Elementlarni** **qo‘yish** **va** **o’chirish**

|  |  |
| --- | --- |
| Operatsiya | Tavsifi |
| s,begin() | Birinchi element uchun ikki yo‘nalishli iteratorni qaytaradi |
| c.end() | Oxirgi element dan keyingi pozitsiya uchun ikki yo‘nalishli  iteratorni qaytaradi |
| c.rbegin() | Teskari saralaganda birinchi element uchun teskari iteratorni  qaytaradi |
| c.rend() | Teskari saralaganda oxirgi element dan keyingi pozitsiya uchun  teskari iteratorni qaytaradi |

1.2.20-jadvalda ro‘yxatlarda elementlarni qo‘yish va o’chirish operatsiyalari keltirilgan. Ro‘yxatlar deklarning hamma funksiyalarni qo‘llaydi, hamda remove() va remove\_if() algoritmlarning maxsus amallashini qo‘llaydi. Odatdagiday, STL qo‘llanganida argumentlarning to‘g‘riligi chaqirish tomonidan ta’minlanadi. Iteratorlar to‘g‘ri pozitsiyaga murojaat qilishi shart, intervalning oxiri boshidan oldin bo‘lmasligi shart, elementlar bo‘sh konteynerdan yo‘qotilmasligi shart.

**1.2.20-jadval.** Ro‘yxatlar uchun qo‘yish va o‘chirish operatsiyalari

Operatsiya

c.insert(pos,elem)

c.insert(pos,n,elem)

Tavsifi

elem nusxasini pos iteratornining pozitsiyasiga qo‘yadi va

yangi elementning pozitsiyasini qaytaradi

elem ning n-nusxalarini pos iteratornining pozitsiyasiga

~~qo‘yadi (qiymatini qaytarmaydi)~~

c.insert(pos,beg,end)

~~keyingi elementning pozitsiyasini qaytaradi~~

yaratiladi)

c.push\_back(e!em) c.pop\_back() c.push\_front(elem) c.pop\_front() c.remove(val)

c.remove\_if(op)

[beg,end) intervalning barcha elementlari nusxasini pos

iteratornining pozitsiyasiga qo‘yadi (qiymatini

elem nusxasini ro‘yxatning oxirisiga qo‘yadi Oxirgi elementni o‘chiradi (uni qaytarmaydi) elem nusxasini ro‘yxatning boshiga qo‘yadi Birinchi elementni o‘chiradi (uni qaytarmaydi) val –qiymatli barcha elementlarni o‘chiradi

op(elem) operatori true qiymatini qaytargan elementlarini

c.erase(pos)

c.erase(beg,end)

c.resize(num)

c.resize(num,elem)

c.clear()

o‘chiradi

pos iteratorning pozitsiyasida elementni o‘chiradi va

keyingi elementning pozitsiyasini qaytaradi [beg,end) intervalidan barcha elementlarni o‘chiradi va

Konteynerni num o‘lchamiga olib keladi (agar bundasize() kattalashsa, yangi elementlar sukutlikda o‘z konstruktori

bilan yaratiladi)

Konteynerni num o‘lchamiga olib keladi (agar bunda size()

kattalashsa, yangi elementlar elem-nusxalari dek

Barcha elementlarni o‘chiradi (konteyner bo‘sh qoladi)

Qo‘yish va o‘chirish tezroq bajariladi, agar elementlar to‘plami ketma-ket

chaqiruvlar bilan emas, bitta chaqiruv bilan bajarilsa.

Elementlarni o‘chirish uchun, ro‘yxatlarda remove() algoritmlarning maxsuslashtirilgan versiyalari mavjud. Ushbu funksiyalar remove() algoritmlaridan tezroq ishlaydi, chunki bular elementlarning o‘rniga faqat ichki ko‘rsatgichlarni qo‘llaydi. Demak, ro‘yxatlar uchun remove() algoritmday emas, sinf funksiyasiday chaqirish kerak. Berilgan qiymat bilan barcha elementlarni o‘chirish uchun, quyidagi tuzilmadan foydalaning:

std::1ist<Elem> coll;

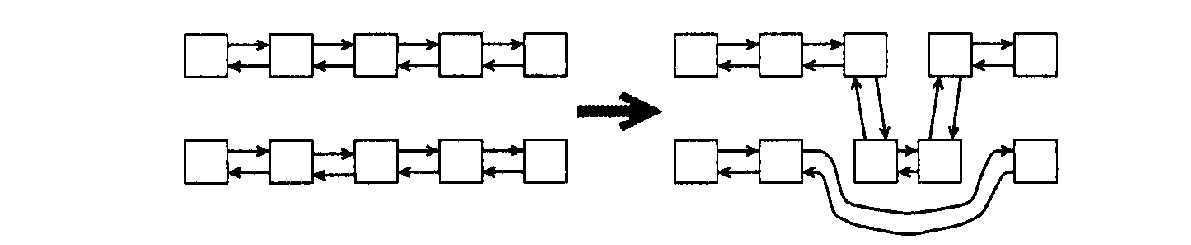
// Qiymatga ega bo‘lgan elementlarni o‘chirish val coll.remove(val);

Ammo, qidirilgan qiymatning birinchi ekzemplyarini o‘chirish uchun,

vektorlar uchun ko‘rsatilgan algoritmdan foydalanishga to‘g‘ri keladi.

remove\_if() funksiyasi elementlarni o‘chirish mezonini aniqlash imkonini beradi, bunisi remove\_if() funksiyasiday yoki fnksiya ob’ektiday bo‘ladi. Ushbu funksiya har bir elementni o‘chiradi, qaysisi uchun uzatilayotgan operatsiya true ni qaytaradi. Misol, bu erda remove\_if() funksiyasi juft qiymatli elementlarni o‘chiradi:

1ist. remove\_if (not 1(bind2nd(modulus<int>().2))); **O’yib** **o‘rnatmoq** **funksiyalari**

Oldinda aytib o‘tilgan, bog‘liq ro‘yxatlarning afzalligi o‘shandaki, ular ixtiyoriy pozitsiyadan vaqtning doimiy qiymati bilan elementlarni qo‘yib va o‘chirib biladi. Ushbu fazilat elementlarni bir konteynerdan ikkinchisiga ko‘chirganda yanada kuchliroq bo‘ladi, chunki ushbu operatsiya bir nechta ichki ko‘rsatgichlarni qayta belgilash bilan o‘tadi (rasm. 1.2.21).

Rasm. 1.2.21. O’yib o‘rnatmoq operatsiyalari ro‘yxat elementlari ketma-

ketligini

o‘zgartiradi.

Ushbu imkoniyatni ta’minlash uchun ro‘yxatlarga qo‘shimcha o‘zgartiradigan funksiyalar zarur bo‘ladi. Bu funksiyalar ketma-ketlikni o‘zgartiradi, elementlarga va intervallarga murojaatlarni qayta bog‘laydi. Ushbu funksiyalar yordamida elementlarni ro‘yxatning ichida, hamda turli ro‘yxatlarning o‘rtasida elementlarni ko‘chirish mumkin (bu ro‘xatlarning tipi bir xil bo‘lishi shart). Funksiyalar ro‘yxati 1.2.22 jadvalida keltirilgan.

**1.2.22** **jadval.** Ro‘yxatlarnio‘zgartirishmaxsusoperatsiyalari

|  |  |
| --- | --- |
| Operatsiya | Tavsifi |
| c.unique() | Dublikatlarni o‘chiradi (bir xil qiymatli elementlar) |

**Istisnolarga** **ishlov** **berish**

|  |  |
| --- | --- |
| c.unique(op) | Dublikatlarni o‘chiradi (bir xil qiymatli elementlar),  qaysilari uchun or qaytaradi true ni |
| cl.splice(pos,c2)    c1.sp1ice( pos,c2, c2 pos) | c2 ning barcha elementlarini cl ga ko‘chiradi iterator  pos pozitsiyasining oldidan  Iteratorning c2pos pozitsiyasidan boshlab, c1  ro‘yxatiga iterator pos pozitsiyasining oldidan c2  elementlarini ko‘chiradi |
| cl.splice(pos,c2,c2beg,c2en d) | [c2beg,c2end) intervalining c2 ro‘yxatining pos  iteratorning pozitsiyasi oldidan c1 ro‘yxatiga  ko‘chiradi (c1 va C2 bir xil bo‘lishi mumkin) |
| c.sort() | Barcha elementlarni < operatori bilan saralaydi |
| c.sort(op) | Barcha elementlarni or mezoni bilan saralaydi |
| d.merge(c2) | C2 barcha elementlarini c1 ga ko‘chiradi, bunda  saralashi saqlanadi (ikala konteyner ham saralgan  elementlarni saqlashi nazarda tutiladi) |
| cl,merge(c2,op) | Barcha elementlarni c2 dan c1 ga ko‘chiradi va or()  bo‘yicha saralashni saqlaydi (or() mezoni bo‘yicha ikala konteyner elementlarining saralanganligi  nazarda tutilayapti) |
| c.reverse() | Barcha elementlarni teskarisiga qo‘yib chiqadi |

STL ning barcha standart konteynerlaridan ro‘yxatlar istisnolarga qaraganda eng mustahkam. Ro‘yxatlarning ustida bajarilgan operatsiyalar deyarli hammasi yoki muvaffaqiyatli bo‘ladi, yoki o‘zgarishni kiritmaydi. Faqatgina sort() funksiyasi va o‘zlashtirish operatorlari. merge(), remove(), remove\_if() va unique() funksiyalari esa, kafolatni shart bilan beradi. Shart quyidagicha: elementlar solishtirilganda, == operatori yoki predikat orqali istisnolar generatsiya bo‘lmasligi lozim. Shunday qilib dasturlash terminologiyasida gapirganda bunday aytish mumkin: ro‘yxatlar *tranzaksion* *xavfsizlikga* ega bo‘ladi, agar o‘zlashtirish operatsiyasi va sort() funksiyasi ishlatilmasa, hamda solishtirilganda istisnolar ishlab chiqmasligi nazorat qilinsa. 1.2.23 jadvalda istinoslarga qarashli maxsus kafolat beruvchi barcha operatsiyalar keltirilgan.

**1.2.23** **jadvali.** Istinoslargaqarashlimaxsuskafolatberuvchi ro‘yxatlaruchunoperatsiya.

kiritmaydi.

~~kiritmaydi.~~

kiritmaydi.

~~solishtirganda generatsiya bo‘lmasa~~

Operatsiya push\_back()

push\_front()

Tavsifi

YOki muvaffaqiyatli tugaydi, yoki o‘zgartirishlarni

YOki muvaffaqiyatli tugaydi, yoki o‘zgartirishlarni

insert() YOkimuvaffaqiyatli tugaydi, yoki o‘zgartirishlarni

kiritmaydi.

pop\_back() Istisnolarni ishlab chiqarmaydi

pop\_front() erase() clear()

resize()

remove()

remove\_if()

unique()

splice()

merge()

reverse()

swap()

Istisnolarni ishlab chiqarmaydi Istisnolarni ishlab chiqarmaydi Istisnolarni ishlab chiqarmaydi

YOki muvaffaqiyatli tugaydi, yoki o‘zgartirishlarni

Istisnolarni ishlab chiqarmaydi, agar ular elementlarni

Istisnolarni ishlab chiqarmaydi, agar ular elementlarni

solishtirganda generatsiya bo‘lmasa Istisnolarni ishlab chiqarmaydi, agar ular elementlarni

solishtirganda generatsiya bo‘lmasa Istisnolarni ishlab chiqmaydi

YOki muvaffaqiyatli tugaydi, yoki o‘zgartirishlarni

kiritmaydi, agar ular elementlarni solishtirganda generatsiya

Istisnolarni ishlab chiqmaydi Istisnolarni ishlab chiqmaydi

**Ro‘yxatni** **ishlatish** **misollari**

Quyida ko‘rsatilgan misolda ro‘yxatlarning maxsus funksiyalarini qo‘llanilishiga e’tibor berish lozim.

// cont/listl.srr #include <iostream> #include <list>

#include <algorithm> using namespace std;

void phntLists (const list<int>& 11. const list<int>& 12) {

cout << "listl:

copy (U.begin(). ll.end(). Ostreara\_terator^-int^cout.'' ")); cout << endl << ”list2: copy (12.begin(). 12.end(), ostrearn\_iterator<int>(cout,” ")): cout << endl << endl; int main()

// Ikkita bo‘sh ro‘yxat yaratish list<int> listl. Iist2:

// Ikala ro‘yxatni elementlar bilan to‘ldirish for (int 1=0; i<6; ++i) { listl.push\_back(i); 1i St2.push\_front(i);

}

printListsdistl, list2);

// listl ning barcha elementlarni birinchi elementdan oldin // 3 qiymati bilan 1ist2ga qo‘yish

// - find() iteratorni 3 qiymati bilan birinchi elementga qaytaradi 1ist2.splise(find(1ist2.begin().1ist2.end(). // qabul pozitsiyasi

3). listl); //manba printLists(1istl. Iist2):

// Birinchi elementni oxiriga qo‘chirish

Iist2.splice(list2.end(),

list2.

Iist2.begin());

printListsdistl, 1ist2);

// qabul pozitsiyasi

// manba

// manba pozitsiyasi

// Ikkinchi ro‘yxatni saralash, o‘zlashtirish listl // va dublikatlarni o’chirish

list2.sort(); listl = 1i st2; list2.unique();

print Lists(1istl, list2);

// birinchi ro‘yxatga ikkita saralangan ro‘yxatni birlashtirish Sliyanie dvux otsortirovannыx spiskov v pervom spiske

listl.merge(list2);

printListsdistl. 1ist2);

}

Dastur quyidagi natijani chiqaradi: listl: 0 12 3 4 5

1ist2: 5 4 3 2 1 0 listl:

1ist2: 5 4 0 1 2 3 4 5 3 2 1 0 listl:

1ist2: 4 0 1 2 3 4 5 3 2 1 0 5 listl: 0 0 1 1 2 2 3 3 4 4 5 5 1ist2: 0 1 2 3 4 5

listl:

0 0 0 1 1 1 2 2 2 3 3 3 4 4 4 5 5 list2:

**Topshiriqlar**

1. Elementni n pozitsiyaga siljitish dasturini tuzing.
2. Ro’yhat nusxasini yarating.
3. Ro’yhat boshiga element qo’yish.
4. Ro’yhatning n-chi elementi o’chirilsin.
5. Ro’yxat n-chi elementidan keyin yangi element qo’yilsin.
6. Ro’yhat elementlari o’sish tartibida joylashtirilsin.
7. Ro’yhat har ikkinchi elementi o’chirilsin.
8. Ro’yhat har uchinchi elementi o’chirilsin.
9. Ro’yhat elementlari kamayish tartibida joylashtirilsin.
10. Ikkita ro’yhat birlashtirilsin.
11. Ikkita ro’yhat umumiy elementlaridan tashkil topgan ro’yhat yaratilsin.
12. Sportchi familiyalaridan tashkil topgan ikkita halqasimon ro’yhat berilgan. Qura tashlash amalga oshirilsin. Birinchi guruhdagi har n-inchi sportchi, ikkinchi guruhdagi har m-inchi sportchi bilan raqib bo’lsin.
13. Lotoreya ishtirokchilari familiyalari va mukofotlar nomlaridan tashkil topgan 2 ta halqasimon ro’yhat berilgan. N ta ishtirokchi g’olib bo’lsin (har K- inchi). Mukofotlarni qayta hisoblash soni - t.
14. O’quvchilar familiyalari va imtihon biletlari raqamlaridan tashkil topgan 2 ta halqasimon ro’yhat berilgan. O’quvchilar tomonidan olingan bilet raqamlari aniqlansin. Imtihon biletlari uchun qayta hisoblash soni - E, o’quvchilar uchun esa
15. K.
16. Mahsulot nomlaridan tashkil topgan ro’yhat berilgan. Ro’yhat elementlaridagi SONY firmasida ishlab chiqilgan mahsulotlardan tashkil topgan yangi ro’yhat yarating.
17. 2 ta guruh talabalari familiyalaridan tashkil topgan 2 ta ro’yhat berilgan. Birinchi guruhdan L ta talaba ikkinchi guruhga o’tkazilsin. Qayta hisoblashlar soni
18. K.
19. 2 ta halqasimon ro’yhatni galma-galdan har 3-elementidan umumiy bitta yangi ro’yhat hosil qiling.
20. 2 ta ro’yhatning bir xil qiymatli elementlaridan yangi halqasimon ro’yhat yarating.
21. 2 ta ro’yhatning bir xil qiymatli elementlarini ro’yhat boshiga o’tkazing.
22. 2 ta ro’yhatning bir xil qiymatli elementlarini ro’yhat oxiriga joylashtiring.

Har bir talaba 4 tadan misol ishlashi kerak.