O’ZBEKISTON RESPUBLIKASI OLIY VA O’RTA MAXSUS
TA’LIM VAZIRLIGI
BUXORO DAVLAT UNIVERSITETI
Fizika –matematika fakulteti
“Axborot texnologiyalari” kafedrasi

Xayrullayeva Gulnoza Shaxrulloyevna

Mavzu: C++ tilida STL kutubxonasidagi dinamik strukturalarni qo’llab quvvatlovchi sinflar bo’yicha o’quv qo’llanma yaratish.

“5111000- Kasb ta`limi(5330200- Informatika va axborot texnologiyalari)”ta’lim yo’nalishi bo’yicha bakalavr
darajasini olish uchun

BITIRUV MALAKAVIY ISHI

“Ish ko’rildi va himoyaga ruxsat berildi”
Kafedra mudiri _____dots.T.B.Boltayev
«___» _____2018 y.

Ilmiy rahbar_____ kata o’qit. I.I.Bakayev
«___» _____2018y.
Taqrizchi_____
«___» _____2018 y.

“Himoya qilishga ruxsat berildi”
Fakultet dekani _____ SH.M.Mirzayev
“_____” _____ 2018 y.

Buxoro-2018

MUNDARIJA

KIRISH

“Yoshlarimizni mustaqil fikrlaydigan, yuksak intellektual va ma’naviy salohiyatga ega bo’lib, dunyo miqyosda o’z tengdoshlariga hech qaysi sohasada bo’sh kelmaydigan insonlar bo’lib kamol topishi, baxtli bo’lishi uchun davlat va jamiyatimizning bor kuch va imkoniyatlarini safarbar etamiz”

Sh. M. Mirziyoyev.

XXI asrda kommunikatsiya tizimining ahamiyati, uning iqtisodiyotni rivojlantirish hamda aholi farovonligini oshirishdagi o’rni barchamizga ma’lum. Birinchi Prezidentimiz I.Karimov O’zbekiston Respublikasi Konstitutsiyasi qabul qilinganligining 18 yilligiga bag’ishlangan tantanali marosimdagi ma’ruzasida quyidagi fikrlarni bildirdi: “Zamonaviy axborot va kompyuter texnologiyalari, raqamli va keng formatli telekommunikatsiyalar, internetni nafaqat maktab, litsey kollej va oliy o’quv yurtlariga, balki har bir oilaga joriy qilish harakatlari bugungi kunda tobora kuchayib bormoqda. Aynan zamonaviy aloqa va axborot texnologiyalari tizimini keng ko’lamda rivojlantirish mamlakatimiz taraqqiy darajasini ko’rsatadigan mezonlardan biri bo’lib xizmat qiladi”

Hozirgi zamonda biror bir sohada ishni boshlash va uni boshqarishni kompyutersiz tasavvur qilish qiyin. XXI asr savodxon kishisi bo’lishi uchun kompyuter savodxoni bo’lish, axborot texnologiyalarini puxta egallamoq lozim. Har bir mutaxassis, u qaysi sohada ishlashdan qat'iy nazar, o’z vazifasini zamon talabi darajasida bajarishi uchun axborotni ishlab chiqaruvchi vositalar va ularni ishlatish uslubiyotini bilishi va ishlash ko’nikmalarga ega bo’lishi zarur.

Respublikamizdagi Oliy va o’rta maxsus ta'lim muasasalarida, Akademik litseylar va kasb-hunar kollejlarida bugungi kunda axborot texnologiyalarini, jumladan Internet tarmog’ini o’quv jarayoniga tadbiq etishni rivojlantirishga katta ahamiyat berilmoqda. Ushbu muammolarni yechimini topib ularni amalda keng qo’llash oliy ta'lim tizimi xodimlari oldiga juda katta vazifalar belgilaydi. Bunda aniq vazifalar sifatida bevosita o’quv jarayonini yaxshilash, o’quv dasturlarini yanada takomillashtirish, o’qitishning zamonaviy pedagogik texnologiyalarini amalga joriy qilish, texnik vositalaridan, Internet tizimidan keng qamrovli foydalanish va pedagogik texnologiyalar asosda ta'lim tizimida interfaol o’qitishni keng joriy qilish masalasi hozirgi davrning dolzarb muammosidir[2].

Prezidentimiz Shavkat Mirziyoyev 2017-yil 15-iyun kuni Toshkentda bo’lib o’tgan “Ijtimoiy barqarorlikni ta’minlash, muqaddas dinimizning sofligini asrash- davr talabi” mavzuidagi anjumanda so’zlagan nutqida yosh avlod tarbiyasi haqida alohida to’xtalib o’tdi. “Bizni hamisha o’ylantirib keladigan yana bir muhim masala-bu yoshlarimizning odob-axloqi, yurish-turishi, bir so’z bilan aytganda, dunyoqarashi bilan bog’liq. Bugun zamon shiddat bilan o’zgaryapti. Bu o’zgarishlarni hammadan ham ko’proq his etadigan kim-yoshlar. Mayli, yoshlar o’z davrining talablari bilan uyg’un bo’lsin. Lekin ayni paytda o’zligini ham unutmasin. Biz kimmiz, qanday ulug’ zotlarning avlodimiz, degan da’vat ularning qalbida doimo aks-sado berib, o’zligiga sodiq qolishga undab tursin. Bunga nimaning hisobidan erishamiz? Tarbiya, tarbiya va faqat tarbiya hisobidan ”, deya ta’kidladi Prezidentimiz.

Bugungi kunda talabalarga sifatli talim berishni tashkil qilishda ilmiy-texnika taraqqiyoti mahsuli bo’lgan zamonaviy axborot texnologiyalari va uning moddiy asosi-kompyuterlar xizmatidan keng foydalanib elektron darslik va qo’llanmalar tashkil etish va internet manbalaridan, hamda masofadan o’qitishning dasturiy vositalaridan foydalanish davr taqozasi bo’lib qolmoqda.

Hozirgi kunda juda ko’p masalalarni bajarishda dinamik ma’lumotlar tuzilmasi orqali bajarishga to’g’ri keladi. Dinamik ma’lumotlar tuzilmasi deganda dastur ishlash jarayonida uning o’lchami o’zgaradigan tuzilma tushuniladi. Bunday

ma’lumotlarning dinamik tuzilmasini ixtiyoriy dasturlash tilida amalga oshirish mumkin. Ushbu BMI da C++ dasturlash tilida dinamik ma’lumotlar tuzilmalarini yaratish va ular ustida bajariladigan amallar to’g’risida ma’lumotlar keltirilgan. Dinamik ma’lumotlar tuzilmalari qanday tasvirlanishi va ular ustida bajariladigan amallar interaktiv o’quv materiallari orqali tushuntirib berilgan.

Mavzuning dolzarbligi. C++ dasturlash tilida ma’lumotlarning bir qancha turlari mavjud bo’lib, ulardan dinamik ma’lumotlar tuzilmasini o’rganish juda murakkab masala hisoblanadi. Chunki juda ko’p masalalarni bajarishda dinamik ma’lumotlar tuzilmasidan foydalansak ham optimal va samarali bo’ladi. Ushbu BMI ning dolzarbligi shundan iboratki dasturchi oldidagi muammolardan biri dinamik xotirani taqsimlanishi hamda katta loyihalarni tayyor namunalar asosida yaratilishini tezlashtiradi. C++ dasturlash tilida dinamik ma’lumotlar tuzilmasini yaratishda ro’yxat, stek, navbatdan foydalaniladi.

Bitiruv malakaviy ishimizning maqsad va vazifalari.

BMI ning maqsadi tayyor standart namunalar shablonidan foydalanishni ochib beruvchi metodik qo’llanma yaratish.

Bitiruv malakaviy ishimizning o’rganilganlik darajasi.

C++ dasturlash tilida STL kutubxonasidagi dinamik strukturalar o’rganib chiqildi (ro’yxat, stek, navbat). O’quv materiallarini tayyorlash jarayoni o’rganildi. O’quv materiallarini tayyorlashda foydalaniladigan dasturiy ta’minotlar o’rganildi va shular asosida o’quv materiallar tayyorlandi.

Bitiruv malakaviy ishimizning predmeti.

C++ dasturlash tili, dinamik strukturalar, o’quv materiallarini tayorlashda foydalaniladigan dasturiy ta’minotlar.

Bitiruv malakaviy ishimizning obyekti sifatida STL kutubxona tanlangan.

Bitiruv malakaviy ishimizning ilmiy farazi dinamik strukturalarni o'rganishni o'quv materiallari orqali amalga oshirish. Bu orqali o'quv jarayonini samaradorligini oshirish.

Bitiruv malakaviy ishimizning yangiligi.

Hozirgi kunda standart namunalar shabloni bo'yicha o'zbek tilidagi o'quv qo'llanmalarning yo'qligidir.

Bitiruv malakaviy ishimizning amaliy ahamiyati shundan iboratki, bu yaratilgan o'quv materiallaridan o'quv jarayonida fanlarni o'tishda foydalanilsa ta'lim oluvchilar mustaqil, erkin va ixtiyoriy bilim olish imkoniyatiga ega bo'ladi, bu esa o'zining yuqori samarasini beradi deb o'ylayman.

Bitiruv malakaviy ishimizning metodologik asosini uning tarkibiy qismi, undagi o'quv va didaktik vositalarning jamlanganligidir. Bu dars jarayonini interfaol tashkil etishda ham amaliy ahamiyat kasb etadi.

Bitiruv malakaviy ishimizning metodlari. C++ dasturlash tilida STL kutubxonasidagi dinamik strukturalarni o'rgatishda eng samarali usul va metodlarni ishlab chiqish va undan o'quv materiallarini tayyorlashda foydalanish.

Bitiruv malakaviy ishining tarkibi va hajmi quyidagicha. Kirish, 1 ta bob, bobning qisqacha xulosasi, adabiyotlar ro'yxati, xotimadan iborat bo'lib jami 100 betga bayon qilingan. I- STL standart namunalar kutubxonasi konteynerlaridan foydalanish bo'yicha ma'lumotlar keltirilgan. Kirish qismi 4 betdan iborat , tushuntirish qismi 90 betdan iborat. Bitiruv malakaviy ishida 5 ta adabiyotdan foydalanildi.

I-BOB. STL STANDART NAMUNALAR KUTUBXONASI KONTEYNEYLARIDAN FOYDALANISH

1.1. STL LAR HAQIDA UMUMIY MA’LUMOT

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 eleentlari 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‘y sinadi. Bir hillari keyingi qismlarda batafsil yoritilgan.

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

Operatsiya	Tavsifi
ContType s	Elementsiz bo‘sh konteynerlarni yaratadi
ContType c(beg,end)	Konteynerni yaratadi va (beg, end) intervalida uni hamma elementlarning nusxalari bilan belgilaydi
s.~ ContType ()	Hamma elementlarni yo‘qotadi va xotirani bo‘shatadi.
c.size()	Aslida bo‘lgan elementlar sonini qaytaradi
c.empty()	Konteynerning bo‘shligini tekshiradi (size()==0ekvivalenti, lekin ba’zan tezroq bajariladi)
c.max_size()	Elementlarning maksimal bo‘ladigan sonini qaytaradi
cl == c2	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
cl.swap(c2)	cl va c2 kontentini joylarini almashtiradi

swap(clrc2)	Xuddi o‘sha global funksiyaning shaklida
c.begin()	Birinchi element uchun iteratorni qaytaradi
c.end()	Iteratorni oxirgi elementdan keyingi pozitsiyaga qaytaradi
c.rbegin()	Orqaga qarab saralaganda, birinchi elementi uchun qaytuvchi iteratorni qaytaradi
c.rend()	Orqaga qarab saralaganda, oxirgi elementdan keyin pozitsiyaga qaytuvchi iteratorni qaytaradi
c.insert(pos,elem)	elem nusxasini qo‘yadi (qaytarilgan qiymatning turli variantlari bilan va birinchi argumentning interpritatsiyasi
c.erase(beg,end)	(beg,end) intervalidan hamma elementlarni yo‘qotadi (ba’zi konteynerlar yo‘qotilgan intervaldan keyin keyingi elementni
c.clear()	Konteynerdan hamma elementlarni yo‘qotadi (konteyner
c.get_allocator()	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::1stream_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<1nt> c(std::istream_iterator<int>(std::cin).

std::istream_iterator<int>()):

Ushbu variantda s *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.

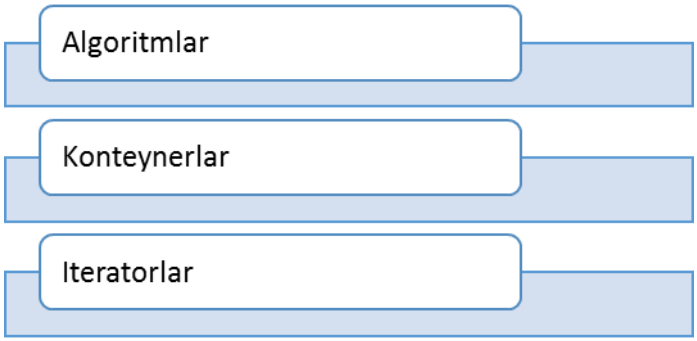
Swap() funksiyasi juda samarali ishlaydi, chunki konteynerlarning faqat ichki berilganlarni qo‘llaydi. Bundan tashqari, Ushbu funksiya ichki ko‘rsatgichlarni taqdim etadi, qaysilari berilganlarga murojaatlarni o‘z ichiga oladi. Demak, swap() funksiyasi o‘zlashtirishda doimo chiqli emas, bir hil murakkablik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



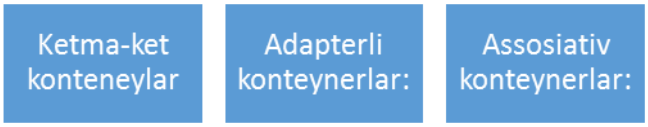
Algoritm nima?

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:



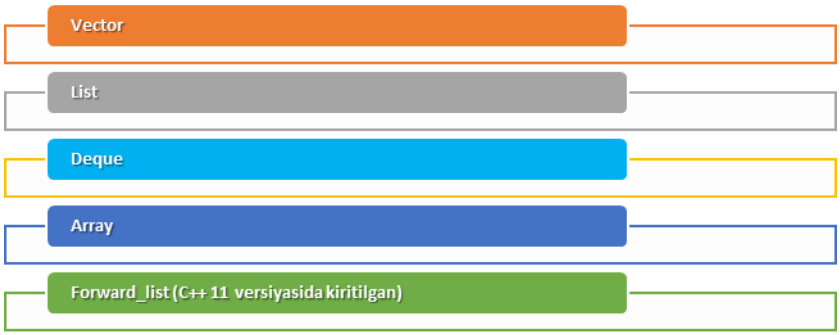
Assosiativ konteynerlar: tez topish mumkin bo’lgan, tartiblangan ma’lumotlar tuzilmasini realizatsiya qilish imkoniyatini beradi.

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.

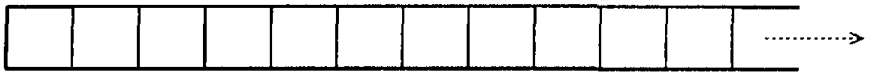
1.2. KETMA-KET KONTEYNERLAR

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.

Vetorlar 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. Vector konstruktorlar va destruktorlari

Operatsiya	Tavsifi
vector<Elem> s	Elementsiz, bo‘sh Vectorni
vector<Elem> cl(c2)	Boshqa Vectorning o‘sha tipli nusxasini yaratadi (Hamma elementlari nusxalanadi)
vector<Elem> s(n)	Jimlik qoidasiga asosan konstruktor orqali, n elementli Vectorni yaratadi
vector<Elem> c(n,elem)	Vectorni yaratadi, qaysisi elem elementining n nusxalari bilan initsializatsiya bo‘ladi
vector<Elem> c(beg, end)	Vectorni yaratadi, qaysisi (beg,end) intervalining 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.	
Operatsiyaya	Tavsifi
c.size()	Asl bo‘lgan elementlar sonini qaytaradi
c.empty()	Konteynerning bo‘shligini tekshiradi (size()==0 ekvivalenti, lekin ba’zan tezroq bajariladi)
c.max_size()	Makismal bo‘ladigan elementlar sonini qaytaradi
Capacity()	Xotirani qayta taqsimlamay, makismal bo‘ladigan elementlar sonini qaytaradi
Reserve()	Agar joriy Vector hajmi berilgandan kichik bo‘lsa, Vector hajmini kattalashtiradi
cl == c2	cl va c2 tengligini tekshiradi
cl! = c2	cl va c2 (ekvivalent !(c1==c2)) tengsizlikni tekshiradi
cl < c2	cl nig c2 dan kichikligini tekshiradi
cl > c2	cl ni c2 dan (ekvivalent c2<c1) kattaligini tekshiradi
cl <= c2	cl ni c2 dan (ekvivalent !(c2<c1)) katta emasligini tekshiradi
cl >= c2	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.

Operatsiya	Tavsifi
cl = c2	cl ga c2 ning Hamma elementlarini o‘zlashtiradi
c.assign(n,elem)	Berilgan elementning n nusxasini o‘zlashtiradi
c.assign(beg,end)	(beg,end) intervalining elementlarini o‘zlashtiradi
cl.5wap(c2)	cl va c2 larning ichidagi narsalarini joylari bilan almashtiradi
swap(cl,c2)	Xuddi o‘sha narsa, faqat global funksiyasi uchun

Hamma o‘zlashtirish vazifalari konstruktorni sukutlitkda chaqiradi, qaysisi 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;        // bo’sh vector!
coll[5] = elem:                 // 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. Iteratorlarni olish operatsiyalari

Operatsiya	Tavsifi
c.begin()	Birinchi element uchun yxtiyoriy ruxsatli iteratorni qaytaradi
c.end()	Oxirgi elementdan keyin pozitsiyasi uchun yxtiyoriy ruxsatli iteratorni
c.rbegin()	Teskari saralaganda birinchi element uchun teskari iteratorni qaytaradi

c.rend()	Teskari saralaganda oxirgi elementdan keyin pozitsiyasi uchun teskari iteratorni qaytaradi

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.

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‘yish va o’chirish operatsiyalari

Operatsiya	Tavsif
c.insert(pos,elem)	Pos iteratorning pozitsiyasiga elem elementning nushasi qo‘yadi va yangi elementning pozitsiyasini qaytaradi
c.insert(pos,n,elem)	Pos iteratorning pozitsiyasiga n nushalarini elem elementning nushasi qo‘yadi (va qiymatni qaytarmaydi)
cinsert(pos,beg,end)	(beg,end) intervalining Hamma elementlarning Pos iteratorning pozitsiyasiga nushasini qo‘yadi (va qiymatni qaytarmaydi)
c.push_back(elem)	elem nusxasini Vectorning oxirisiga qo‘shadi
c.pop_back()	oxirgi elementni yo‘qotadi (uni qaytarmaydi)
c.erase(pos)	Pos iteratorning pozitsiyasida elementni yo‘qotadi va keyingi elementning pozitsiyasini qaytaradi
c.erase(beg,end)	(beg,end) intervalining Hamma elementlarini yo‘qotadi va keyingi elementning pozitsiyasini qaytaradi

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

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

- 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().

ostreamjterator<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 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> maxsus operatsiyalari

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 bitni belgilash)
m[idx1] = m[idx2]	idx2 indekli elementga idx1 indeksli elementning qiymatini o‘zlashtiradi

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.

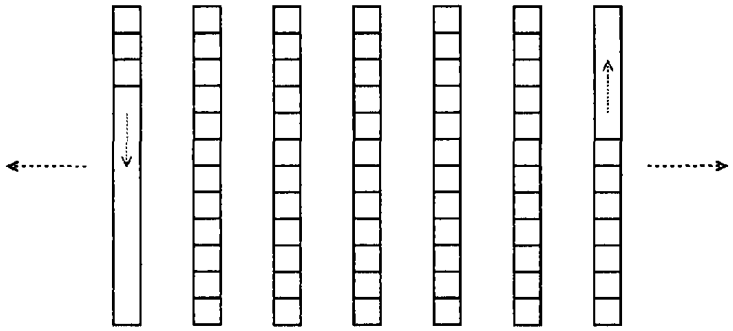
reference ichki tipi faqatgina nokonstanatali vector<bool> tipli konteynerlari uchun qo‘llaniladi. Konstantali elementlarga murojaat funksiyalar oddiy bool tipli qiymatlarni qaytaradi.

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 va destruktorlari

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
c≈vdeque<Elem>()	Hamma elementlarni yo’qotib, xotirani bo‘shatadi

1.2.12-jadval. Deklar ustida o‘zgartimaydigan operatsiyalar.

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. Deklar ustida o‘zgartiradigan operatsiyalari

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
c.insert(pos,n,e1em)	Elem elementning n – nuxalarini iteratorning pos pozitsiyasiga qo‘yadi (qiymatini qaytarmaydi)
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)

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
c.resize(num,elem)	Konteynerni num o'lchamiga keltiradi (agar bunda size() kattalashsa, yangi elementlar elem nusxalari dek yaratiladi)
c.clear()	Hamma elementlarni yo'qotadi (konteyner bo'sh qoladi)

Deklar operatsiyalari vektorlar operatsiyalaridan faqat quyidagi nisbatlarda 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> finclude <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

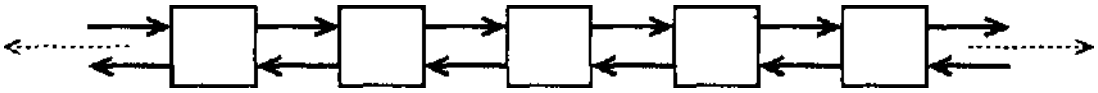
string

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
list<Elem> s	Elementsiz, bo‘sh ro‘yxatni yaratadi
list<Elem> d(c2)	O‘sha tipli boshqa ro‘yxatni nusxasini yaratadi (xamma elementlar nusxalanadi)
list<Elem> s(n)	N- elementli ro‘yxatni tuzadi, qaysisi sukutlikda konstruktor orqali yaratiladi
list<Elem> c(n,elem)	elem elementininig n-nusxalari bilan initsializatsiya bo‘ladigan ro‘yxatni yaratadi
list<Elem> c(beg,end)	[beg,end) interva elementlari bilan initsializatsiya bo‘ladigan ro‘yxatni yaratadi
c.~list<Elem>()	Hamma elementlarni yo‘qotib xotirani 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‘yxatlar uchun 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‘xatning elementlariga to‘g‘ridan-to‘g‘ri murojaat operatsiyalari

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 (Icoll .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. Iteratorlarni olish operatsiyalari

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

Elementlarni qo‘yish va o’chirish

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	Tavsifi
c.insert(pos,elem)	elem nusxasini pos iteratornining pozitsiyasiga qo‘yadi va yangi elementning pozitsiyasini qaytaradi
c.insert(pos,n,elem)	elem ning n-nusxalarini pos iteratornining pozitsiyasiga qo‘yadi (qiymatini qaytarmaydi)

c.insert(pos,beg,end)	[beg,end) intervalning barcha elementlari nusxasini pos iteratornining pozitsiyasiga qo‘yadi (qiymatini
c.push_back(e!em)	elem nusxasini ro‘yxatning oxirisiga qo‘yadi
c.pop_back()	Oxirgi elementni o‘chiradi (uni qaytarmaydi)
c.push_front(elem)	elem nusxasini ro‘yxatning boshiga qo‘yadi
c.pop_front()	Birinchi elementni o‘chiradi (uni qaytarmaydi)
c.remove(val)	val –qiymatli barcha elementlarni o‘chiradi
c.remove_if(op)	op(elem) operatori true qiymatini qaytargan elementlarini o‘chiradi
c.erase(pos)	pos iteratorning pozitsiyasida elementni o‘chiradi va keyingi elementning pozitsiyasini qaytaradi
c.erase(beg,end)	[beg,end) intervalidan barcha elementlarni o‘chiradi va keyingi elementning pozitsiyasini qaytaradi
c.resize(num)	Konteynerni num o‘lchamiga olib keladi (agar bunda size() kattalashsa, yangi elementlar sukutlikda o‘z konstruktori bilan yaratiladi)
c.resize(num,elem)	Konteynerni num o‘lchamiga olib keladi (agar bunda size() kattalashsa, yangi elementlar elem-nusxalari dek yaratiladi)
c.clear()	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,

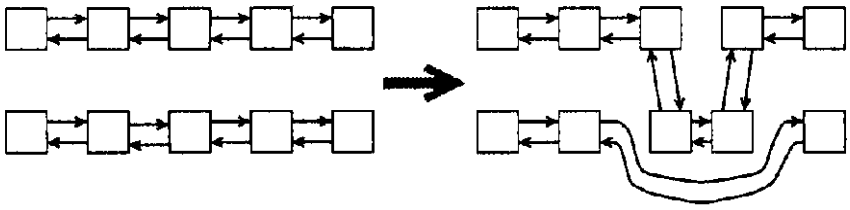
vektorlar uchun ko‘rsatilgan algoritmdan foydalanishga to‘g‘ri keladi.

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‘yxatlarni o‘zgartirish maxsus operatsiyalari

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

c.unique(op)	Dublikatlarni o‘chiradi (bir xil qiymatli elementlar), qaysilari uchun or qaytaradi true ni
cl.splice(pos,c2)	c2 ning barcha elementlarini cl ga ko‘chiradi iterator pos pozitsiyasining oldidan
c1.sp1ice(pos,c2, c2 pos)	Iteratorning c2pos pozitsiyasidan boshlab, c1 ro‘yxatiga iterator pos pozitsiyasining oldidan c2 elementlarini ko‘chiradi
cl.splice(pos,c2,c2beg,c2end)	[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

Istisnolarga ishlov berish

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. Istinoslarga qarashli maxsus kafolat beruvchi ro‘yxatlar uchun operatsiya.

Operatsiya	Tavsifi
push_back()	YOki muvaffaqiyatli tugaydi, yoki o‘zgartirishlarni kiritmaydi
push_front()	YOki muvaffaqiyatli tugaydi, yoki o‘zgartirishlarni kiritmaydi
insert()	YOki muvaffaqiyatli tugaydi, yoki o‘zgartirishlarni kiritmaydi.
pop_back()	Istisnolarni ishlab chiqarmaydi
pop_front()	Istisnolarni ishlab chiqarmaydi
erase()	Istisnolarni ishlab chiqarmaydi
clear()	Istisnolarni ishlab chiqarmaydi
resize()	YOki muvaffaqiyatli tugaydi, yoki o‘zgartirishlarni kiritmaydi
remove()	Istisnolarni ishlab chiqarmaydi, agar ular elementlarni solishtirganda generatsiya bo‘lmasa
remove_if()	Istisnolarni ishlab chiqarmaydi, agar ular elementlarni solishtirganda generatsiya bo‘lmasa
unique()	Istisnolarni ishlab chiqarmaydi, agar ular elementlarni solishtirganda generatsiya bo‘lmasa
splice()	Istisnolarni ishlab chiqmaydi
merge()	YOki muvaffaqiyatli tugaydi, yoki o‘zgartirishlarni kiritmaydi, agar ular elementlarni solishtirganda generatsiya
reverse()	Istisnolarni ishlab chiqmaydi
swap()	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(). ostrearajterator^-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(),              // qabul pozitsiyasi
                        list2.                 // manba
        Iist2.begin());                        // manba pozitsiyasi
        printListsdistl, 1ist2);
// 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
otsortirovannых 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:
```

1.3. ADAPTERLI KONTEYNERLAR

C ++ standart kutubxonasi STLga kiradigan konteynerlar bilan cheklanmaydi. Bundan tashqari, maxsus maqsadlar uchun mo'ljallangan va oddiy, deyarli aniq interfeyslarga ega konteynerlar ham mavjud. Bunday konteynerlarni guruhlarga bo'lish mumkin.

Konteynerli adapterlar deb ataladigan. Ushbu guruhga standart STL konteynerlarini maxsus maqsadlar uchun moslashtirilgan konteynerlar kiradi. Uchta standart konteyner adapteri mavjud:

□ staklar;

□ navbat;

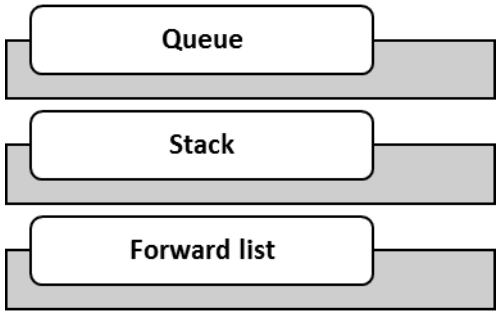
□ ustuvor navbat.

Ustuvor navbatlar elementlarga belgilangan tartib-qoidalarga ko'ra avtomatik ravishda tartiblangan navbatlar deb ataladi. Shunday qilib, navbatdagi "keyingi" elementning qiymati "oldingi" qiymatdan "katta" dir.

Maxsus konteyner bitset.

Bitset konteynerlari ixtiyoriy soni, lekin qattiq sonli bitga ega bo'lgan bit maydondir. Standart C ++ kutubxonasiga qiymatlar uchun o'zgaruvchan o'lchamdagi vektor <bool> maxsus konteynerni ham o'z ichiga oladi.

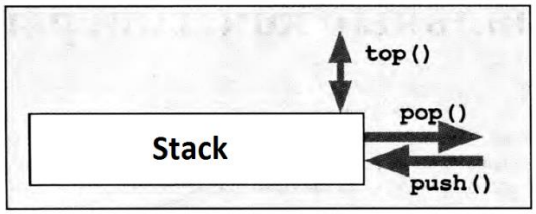
Adapterli konteynerlar: ketma-ket konteneylar uchun boshqa interfeysni taqdim etadi.



Steklar

Stack() sinfi "oxirgi kelganga birinchi xizmat" (LIFO) tamoyili asosida ishlaydigan steckni amalga oshiradi. Push() funktsiyasi stekga elementlarning

ixtiyoriy sonini qo'shib qo'yadi (1.3.1-rasmga qarang), pop() funktsiyasi esa elementlarni teskari tartibda o'chirib tashlaydi.



1.3.1-rasm. Stek interfeysi.

Stekni dasturda ishlatish uchun dasturga <stack> sarlavha faylini kiritish kerak:

#include <stack>

<Stack> faylida stack sinfi quyidagicha tavsiflanadi:

```
namespace std {
    template <class T,
              class Container = deque<T> >
    class stack;
}
```

Shablonning birinchi parametri elementlarning turini belgilaydi. Shart bo'lmagan ikkinchi shablon parametri ma'lumotlar saqlash uchun ichki dastur tomonidan ishlatiladigan konteynerni bildiradi. Aslida sukutlikda bu - dek. Bunisi bunday tushuniladi: deklar(vektorlardan ko’ra) elementlarni yo'q qilganda xotirani bo'shatadi va xotirani qayta taqsimlaganda barcha elementlarni nusxalamaydi.

Masalan, quyidagi e’lon int tipidagi elementlarni belgilaydi:

std::stack<int> st: // Butun sonlar steki

Stekni qo'llashi: stek bilan bo’lgan operatsiylarni qo’llanilgan konteyner bilan muvofiq bo’lgan operatsiyalarga aks ettiradi. (1.3.2-rasm). back (), push_back () va pop_back () funktsiyalari bilan qo'llab-quvvatlanadigan har qanday ketma-ket konteynerdan foydalanishingiz mumkin. Masalan, stek elementlari vektorda yoki ro'yxatda saqlanishi mumkin:

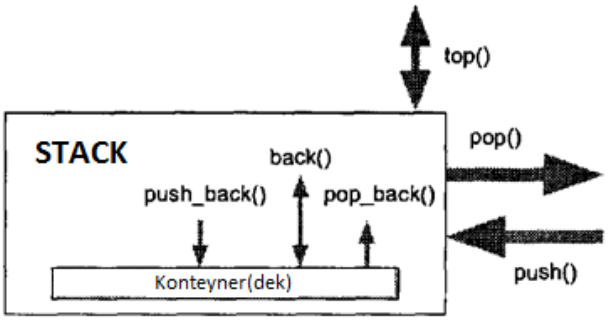
std::stack<int.std::vector<int> > st; // vector asosida butun sonlarning steki

Asosiy interfeys

Asosiy stack interfeysi push(), top() va pop() funktsiyalaridan iborat:

- Push() funktsiyasi stekga qo'yilgan elementni qo'shadi;

- top() funksiyasi stekning yuqori elementini qaytaradi;
- Pop() funktsiyasistekdan elementni o’chiradi.



Rasm-1.3.2. Stekning ichki interfeysi.

Eslatma: pop() funktsiyasi yuqori elementni o'chirib tashlaydi, lekin uni qaytarmaydi, top() funktsiyasi esa stekning yuqori elementini o’chirmay qaytaradi. Shuning uchun, yuqori elementni qayta ishlash va uni to'plamdan olib tashlash uchun har ikkala funksiyani ham chaqirish kerak bo’ladi. Bunday interfeys biroz noqulay, lekin ishlov bermasdan yuqori elementni o’chirishda, yanada samarali bo'ladi. Agar stekda hech qanday element bo'lmasa, yuqoridagi top() va pop() funktsiyalarining harakati aniqlanmagan. Stekda elementlarning mavjudligi size() va empty() funktsiyalari tomonidan tekshiriladi.

Standart stack() interfeysi sizga mos kelmasa, qulayroq interfeyslarni osongina yozishingiz mumkin.

Stekni ishlatish misoli

stack() sifini ishlatilishi :

```
// cont/stack1.cpp
#include <iostream>
#include <stack>
using namespace std;

int main()
{
    stack<int> st;
    //stackga 3 ta element kiritish
    st.push(1);
    st.push(2);
    st.push(3);
     //stackdan 2 elementni o'chirish va chiqarish
    cout << st.top() << ' ';
    st.pop();
    cout << st.top() << ' ';
    st.pop();
     //yuqoridagi elementni  o'zgartirish
    st.top() = 77;

  //2 ta yangi element qo'shish
   st.push(4);
   st.push(5);
    //o'zgartirishsiz elementni o'chirish
   st.pop();
  //qolgan elementlarni ekranga chiqarish va o'chirish
   while (!st.empty()) {
        cout << st.top() << ' ';
        st.pop();
   }
   cout << endl;
}
```

Dastur bajarilishining natijasi:

3 2 4 77

STACK sinfining tuzilishi

stack() sinfi interfeysi juda kichik, uni odatdagi dasturni tahlil qilish yo'li bilan osongina tushunish mumkin:

```
namespace std {
    template <class T, class Container = deque<T> >
    class stack {
      public:
        typedef typename Container::value_type value_type;
        typedef typename Container::size_type  size_type;
        typedef          Container             container_type;
      protected:
        Container c;   // Контейнер
      public:
        explicit stack(const Container& = Container());

        bool        empty() const             { return c.empty(); }
        size_type   size()  const             { return c.size(); }
        void        push(const value_type& x) { c.push_back(x); }
        void        pop()                     { c.pop_back(); }
        value_type& top()                     { return c.back(); }
        const value_type& top() const         { return c.back(); }
    };

    template <class T, class Container>
      bool operator==(const stack<T, Container>&,
                      const stack<T, Container>&);
    template <class T, class Container>
      bool operator< (const stack<T, Container>&,
                      const stack<T, Container>&);

                    // (Solishtirish boshqa operatorlari) }
```

Quyida stack() sinf a'zolarining batafsil tavsifi keltirilgan.

Tiplarning ta’riflanishi

Stek ::value_type

- Elementlar tipi.
- Ekvivalenti:

konteyner::value_type *stek:*:size_type

- O’lcham uchun belgisiz butun tip.
- Ekvivalent:

konteyner::size_type

stek::container_type

konteyner tipi.

Operatsiyalar

*stek::*stack ()

- Sukutlikda konstruktor.
- Bo’sh stekni yaratadi.

explicit stek < ::stack (const Containers *cont)*

- *cont* elementlari bilan initsializatsiya bo’lgan stekni yaratadi
- *cont* barcha elementlari stekga nusxalanadi

size_type *stek::*size () const

- Joriy elementlar sonini qaytaradi.
- Stekda elementlarning yo'qligini tekshirish uchun empty() funktsiyasidan foydalanish tavsiya etiladi, chunki u tez ishlaydi.

bool *stek:*:empty ( ) const

- Stekning bo'shligini tekshiradi.
- Ekvivalent(lekin tez ishlashi mumkin):

stek: :size()==0

void *stek::*push (const *value_type& elem)*

- *Elem* ning nusxasini stekga kiritadi, natijada u yangi birinchi elementga aylanadi.

*value_type& stek::*top( )

const *valuejiypeb, stek::*top( ) const

- Ikkala shakllar ham stekning yuqori elementini qaytaradi, ya'ni oxirgi kiritilgan elementni (barcha boshqa elementlardan keyin) qaytaradi.
- Chaqirishdan oldin, stek kamida bitta element (size( )> 0) mavjudligiga ishonch hosil qilishingiz kerak, aks holda qo'ng'iroq chaqirilishning oldindan aniqlanmaydigan oqibatlariga olib keladi.
- Birinchi shakl konstantasiz steklar uchun murojaatni qaytaradi, bunisi stekdagi yuqori elementni o'zgartirishga imkon beradi. Bunisi yaxshimi yoki yo'qmi, o'zingni tanlang.
- void *stek*::ror( )
- Stekdan yuqori elementni, ya'ni oxirgi kiritilgan elementni (barcha boshqa elementlardan keyin) o'chiradi.
- Funktsiyaning qaytish qiymati yo'q. Yuqori elementning qiymatini qayta ishlash uchun avval top() funktsiyasini chaqirishingiz kerak.
- Chaqirishdan oldin, stek kamida bitta element (size( )> 0) mavjudligiga

ishonch hosil qilishingiz kerak, aks holda chaqiruv kutilmagan holatga olib keladi.

- bool *taqqoslash* (const sgek& *stackl.* const *stek& stack2*)
- bir xil turdagi ikkita steklarni taqqoslash natijasini qaytaradi.
- *taqqoslash* parametri quyidagi operatsiyalardan biridir:
- operator == operator != operator < operator > operator <= operator >=
- Steklar bir-biriga teng hisoblanadi, agar ularda elementlar soni bir xil bo'lsa, agar elementlarning juftligi mos kelsa va bir hil ketma-ketligda bo'lsa (ya'ni tenglik uchun ikkita mos keladigan elementni tekshiruvchi har doim true ni beradi).
- Konteynerlar orasidagi "katta/kichik" nisbati leksik mezonlarga muvofiq tekshiriladi, leksikografiya mezonlari leksicographical_cornpare() algoritmining tavsifida ko'rib chiqiladi.

Stekning amallash dasturi

Standart stack() sinfi birinchi navbatda qulaylik va xavfsizlikni emas, balki tezligini beradi. Sizning e'tiboringiz steklarni nostandart amalga oshirishi berilgan. Bunisi ikki afzalliklarga ega:

- pop() funktsiyasi yuqori elementni qaytaradi;
- Bo'sh stekda pop() va top() funksiyalari istisnolarni yaratadi.

Bundan tashqari, odatdagi stack foydalanuvchilari uchun kerak bo'lmagan operatsiyalar (masalan, taqqoslash operatsiyalari) bajarilishdan chiqarildi. Olingan stek sinfi quyida ko'rsatilgan.

```
// cont/Stack.hpp
/* ************************************************************
 *
 *   //stack ning qulay va xavfsiz sinfi tasvirlangan
 * ************************************************************/
#ifndef STACK_HPP
#define STACK_HPP

#include <deque>
#include <exception>

template <class T>
class Stack {
  protected:
    std::deque<T> c;          //Elementlarni saqlash uchun konteyner

  public:
    /* Bo'sh stackda pop() va top() funksiyalar orqali generatsiya
     * qilinadigan sinf istisnosi
     */
    class ReadEmptyStack : public std::exception {
      public:
        virtual const char* what() const throw() {
            return "read empty stack";
        }
    };
```

// elementlar miqdori

```
typename std::deque<T>::size_type size() const {
    return c.size();
}
```

//stack bo'shlikga tekshirish

```
bool empty() const {
    return c.empty();
}
```

//stackga elementni qo'shish

```
void push (const T& elem) {
    c.push_back(elem);
}
```

//Stack qiymatini qaytarish

```
T pop () {
    if (c.empty()) {
        throw ReadEmptyStack();
    }
```

```
        T elem(c.back());
        c.pop_back();
        return elem;
    }

    // Yuqoridagi elementni olish
    T& top () {
        if (c.empty()) {
            throw ReadEmptyStack();
        }
        return c.back();
    }
};

#endif /* STACK_HPP */
```

Stekning ushbu sinf ishlatganda, avvalgi namuna quyidagi shaklda yozilishi mumkin:

```
// cont/stack2.cpp
#include <iostream>
#include "Stack.hpp"        // Nostandart stack sinfidan foydalanish
using namespace std;

int main()
{
   try {
      Stack<int> st;

      // stack 3 ta element kiritish
      st.push(1);
      st.push(2);
      st.push(3);

      // stackdan 2 ta elementni chiqarish
      cout << st.pop() << ' ';
      cout << st.pop() << ' ';

      // Yuqoridagi elementni o'zgartirish
      st.top() = 77;

      // 2 ta element qo'shish
      st.push(4);
      st.push(5);

      // O'zgartirishsiz elementni chiqarish
      st.pop();

/* elementlarni ekranga chiqarish
 * - xatolik yuz beradi 1 ta element yetishmaydi
       */
      cout << st.pop() << ' ';
      cout << st.pop() << endl;
      cout << st.pop() << endl;
   }
   catch (const exception& e) {
      cerr << "EXCEPTION: " << e.what() << endl;
   }
}
```

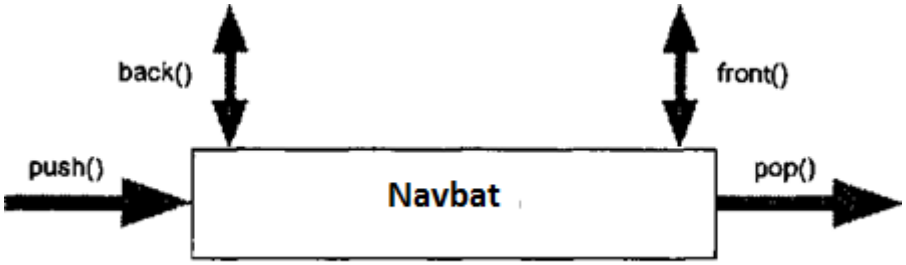
Pop() ning so'nggi chaqiriqida ataylab xato qilib qo'ydik. Stekning standart sinfidan ko’ra, sinfimiz istisno ishlab chiqaradi. Dasturning ishlashi natijasi quyidagich bo’ladi:

3 2 4 77

EXCEPTION: read empty stack

Navbatlar

Quyi sinfi “birinchi yetib kelgan, birinchi bo’lib xizmat ko’rsatadi” degan prinsip asosida ishlab navbatni faollashtiradi.(FIFO). Push() funksiyasi elementlarni navbatga klassik buffer obmen tarzida ko’rib chiqishi mumkin (1.3.3-rasm)



1.3.3-rasm. Navbat interfeysi

Navbatni dasturda ishlatish uchun <queue> nomli nomli faylni qo’shishi kerak. <queue> faylni quyidagi tarzda aniqlanadi.

#include <queue>

<Queue> faylida navbat navbati quyidagicha tavsiflanadi:

```
namespace std {
    template <class T,
              class Container = deque<T> >
    class queue;
}
```

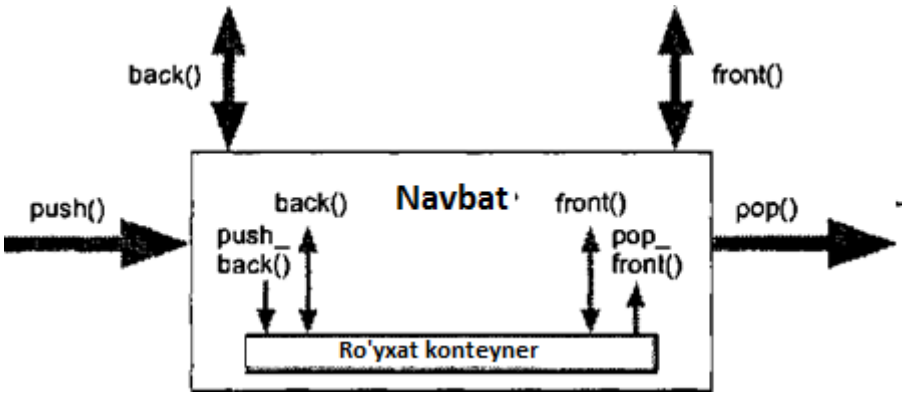
Birinchi shablon element tipini aniqlaydi. Shartsiz ikkinchi shablon sozlamasi ichki amallashda elementni saqlash uchun mo’ljallangan konteynerni aniqlaydi. Bu sukutlikda dek.

Misol uchun, navbat e’loni satr elementlari bilan aniqlaydi:

std::queue<std::string> buffer: // satrli navbat

Navbatni amalga oshirish jarayoni navbat bilan operatsiyalarni ishlatiladigan konteynerning mos operatsiyalarga aks etadi (1.3.4-rasmga qarang). Front(), back(), push_back() va pop_front() uchun qo'llab-quvvatlanadigan ketma-ket konteynerlarning har qanday sinfidan foydalanishingiz mumkin. Masalan, navbatdagi elementlar ro'yxatda saqlanishi mumkin:

std::queue<std::string.std::list<std::string> > buffer;



1.3.4-rasm. Interfeysning ichki tuzilmasi.

Asosiy interfeys

Asosiy navbat interfeysi push(), front(), back() va pop () funktsiyalaridan iborat. Push() funktsiyasi elemntni navbatga qo'yadi;

- Front() funksiyasi navbatdagi elementni elementni qaytaradi (boshqasidan oldin kiritilgan);
- Back() funksiyasi navbatning oxirgi elementini qaytaradi (boshqalardan kechroq kiritilgan);
- pop() funktsiyasi elementni navbatdan o'chiradi.

Eslatma: pop() funktsiyasi keyingi elementni o'chirib yuboradi, lekin qaytarmaydi, front() va back() funktsiyalari esa keyingi elementni o'chirmasdan qaytaradi. Shuning uchun, keyingi elementni boshqarish va uni navbatdan olib tashlash uchun har doim front() va pop() funktsiyalarini chaqirishingiz kerak. Bunday interfeys biroz noqulay, lekin keyingi elementni qayta ishlamay turib o'chirish samarali bo'ladi. Navbatda biron bir element bo'lmasa, front(), back() va pop() funksiyalarining harakati aniqlanmagan. Navbatda elementlarning mavjudligi size() va empty()funktsiyalari bilan tekshiriladi.

Navbatni ishlatilishi misoli

queue() sinfning navbatini ishlatishning misoli:

```
// cont/queue1.cpp
#include <iostream>
#include <queue>
#include <string>
using namespace std;

int main()
{
    queue<string> q;

    //  Navbatga 3 ta element qo'shish
    q.push("These ");
    q.push("are ");
    q.push("more than ");

//  Navbatdan 2 ta elementni chiqarish va o'chirish
cout << q.front();
q.pop();
cout << q.front();
q.pop();

//  2 ta elementni qo'shish
q.push("four ");
q.push("words!");

// 1-ta element o'chirish
q.pop();

 //  2 ta elementni chiqarish va o'chirish
 cout << q.front();
 q.pop();
 cout << q.front() << endl;
 q.pop();

 //  Navbatning elementlar miqdorini chiqarish
 cout << "number of elements in the queue: " << q.size()
      << endl;
}
```

Dasturning bajarilish natijasi quyidagiday bo’ladi:

These are four words!

number of elements in the queue: 0

QUEUE sinfning tuzilishi

queue() sinfini oddiy amallanilishi singari stack() sinfining qo'llanilishi uchun hech

qanday maxsus izoh talab qilinmaydi:

```
namespace std {
    template <class T, class Container = deque<T> >
    class queue {
      public:
        typedef typename Container::value_type value_type;
        typedef typename Container::size_type  size_type;
        typedef          Container             container_type;
      protected:
        Container c;   // Контейнер
      public:
        explicit queue(const Container& = Container());

        bool        empty() const             { return c.empty(); }
        size_type   size()  const             { return c.size(); }
        void        push(const value_type& x) { c.push_back(x); }
        void        pop()                     { c.pop_front(); }
        value_type& front()                   { return c.front(); }
        const value_type& front() const       { return c.front(); }
        value_type& back()                    { return c.back(); }
        const value_type& back() const        { return c.back(); }
    };

    template <class T, class Container>
      bool operator==(const queue<T, Container>&,
                      const queue<T, Container>&);
    template <class T, class Container>
      bool operator< (const queue<T, Container>&,
                      const queue<T, Container>&);
    ... // (Другие операторы сравнения)
}
```

Quyida queue() sinfning navbatdagi a'zolarining batafsil tavsiflari keltirilgan.

Tiplarni aniqlash

navbat ::value_type

- Elementlar tipi.
- Ekvivalenti:

konteyner::value_type

navbat ::size_type

- O’lchov qimatlari uchun belgisiz butun tipi.
- Ekvvalenti:

konteyner ::size_type

navbat ::container_type

Konteyner tipi.

Operatsiyalar

Navbat ::queue()

- Sukutlikda konstruktor.
- Bo’sh navbatni yasaydi.

explicit *navbat* ::queue (const Container& *cont)*

- *cont* elementlari bilan initsializatsiya qilingan navbatni yaratadi.
- *Cont* barcha elemetlari navbatga nusxalanadi.

size_type *navbat*: :size () const()

- Joriy elementlarning sonini qaytaradi.
- Navbatdagi yo'q bo’lgan elementlarni tekshirish uchun empty() funktsiyasidan foydalanish tavsiya etiladi, chunki u tezroq ishlashi mumkin.

bool *navbat* ::empty() const

- Navbatning bo’shligini tekshiradi.
- Ekvivalent (lekin tezroq ishlashi mumkin):

Navbat::size()==0

void *navbat:*:push (const *value_type& elem)*

n navbat *elem* parametri elementlari nusxasini qo’yadi, natijada u yangi oxirgi elementga aylanadi.

value_type& ochered :: front()

const *value_type& navbat* :: front() const

- Ikkala shakllar navbatdagi elementni qaytaradi, ya'ni birinchi navbatda kiritilgan element (navbatdagi boshqa elementlardan keyin).
- Chaqirishdan oldin, navbat kamida bitta elementni (size()> 0) o'z ichiga olishi kerak, aks holda chaqiruv kutilmagan holatga olib keladi.
- Nokonstant navbatlar uchun birinchi shakli murojaatni qaytaradi, bu navbatdagi elementni o'zgartirishga imkon beradi. Bunisi yaxshimi yoki yo'qmi, o'zingni tanlang.

Value_type& *navbat* back()

const *va1ue_type& navbat*:: back() const

- Ikkala shakllar navbatdagi oxirgi elementni qaytaradi, ya'ni so’nggi navbatda kiritilgan element (navbatdagi boshqa elementlardan keyin).
- Chaqirishdan oldin, navbat kamida bitta elementni (size()> 0) o'z ichiga olishi kerak, aks holda chaqiruv kutilmagan holatga olib keladi.
- Nokonstant navbatlar uchun birinchi shakl murojaatni qaytaradi, bu navbatdagi oxirgi elementni o'zgartirishga imkon beradi. Bunisi yaxshimi yoki yo'qmi, o'zingni tanlang.

void *navbat* :: pop()

- navbatdan navbatdagi elementni, ya'ni birinchi bo'lib kiritilgan elementni (navbatning barcha elementlaridan oldin) o'chiradi.
- Funktsiyaning qaytish qiymati yo'q. Yuqori elementning qiymatini qayta ishlash uchun avvalo front() funktsiyasini chaqirish kerak.
- Chaqirishdan oldin, navbat kamida bitta elementni (size()> 0) o'z ichiga olishi kerak, aks holda chaqiruv kutilmagan holatga olib keladi.
- bool *taqqoslash* (const *navbat*& *queue 1,* const *navbat*& *queue2)*
- Ikkita bir hil tipli nabatlarning taqqoslash qiymatini qaytaradi.
- *Taqqoslash* parametri — keyingi operatsiyalardan birisi:
- operator = operator != operator < operator > operator <= operator >=
- Navbatlar bir-biriga teng hisoblanadi, agar ularda elementlar soni bir xil bo’lsa, agar elementlarning juftligi mos kelsa va bir hil ketma-ketligda bo’lsa (ya'ni tenglik uchun ikkita mos keladigan elementni tekshiruvchi har doim true ni beradi).
- Konteynerlar orasidagi "katta/kichik" nisbati leksik mezonlarga muvofiq tekshiriladi, leksikografiya mezonlari lexicographical_compare() algoritmining tavsifida ko'rib chiqiladi.

Navbatni amallash dasturi

Foydalanuvchi queue() sinfi birinchi navbatda qulaylik va xavfsizlikni emas, balki tezlikni keltirib chiqaradi. Navbatning nostandart dasturga e'tibor bering. Ush bu dastur ikkita afzalliklarga ega:

- ror() funktsiyasi keying elementni qaytaradi;

- navbat bo’sh bo’lganda pop() va front()funktsiyalari istisnolarni ishlab chiqaradi.

XULOSA

Bitiruv malakaviy ishda C++ ning standart shablonlar kutubxonasi bo'yicha o'quv tayyorladim. O'quv qo'llanma tayyorlash jarayonida C++ standart shablon kutubxonasi bo'yicha ilmiy ish olib borgan Джосаттис, Вандевурд kabi olimlarning kitoblari bilan tanishishib chiqdim. C++ standart shablonlar kutubxonasi dasturchiga tayyor konteynerlardan foydalanish imkoniyatini beradi. Konteynerlar asosiy vazifasi dinamik xotirani kerakli vaqtda belgilash va o'chirishdan iborat. Dasturchi dastruning mohiyatiga qarab konteynerlar toifasini hamda turini tanlaydi. Bitiruv malakaviy ishda ketma-ket konteynerlar (vector, deque, list), adapterli konteyner yoki maxsus konteynerlar (stack, queue, forward_list), assosiativ konteynerlar (set va multiset, xarita va multi xaritalar) kabilar haqida ma'lumot to'plandi hamda bir tizimga keltirildi. Bundan tashqari bitiruv malakaviy ish natijasi sifatida o'quv qo'llanmada konteynerlarning o'xshash hamda farqli tomonlari ko'rsatib o'tilgan. O'quv qo'llanmada keltirilgan kodlar Microsoft Visual Studio 2015 update3, Dev C++, Code Blocks dasturlash muhitlarida tekshirildi.

Bitiruv malakaviy ishni tayyorlash vaqtida konteyner, iteratorlar, algoritmlar kabi tushunchalarni qanday foydalanish haqida ma'lumotga ega bo'ldim.

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.

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.

Konteynerlar yoki sinf konteynerlari ma'lumot va obyektlarni saqlaydi. Interatorlar kolleksiya obyektlarining elementlari orqali o'tish uchun ishlatiladi.

Bu kolleksiyalar konteynerlar yoki to’plam osti konteynerlar bo’lishi mumkin. 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.

Xulosa qilib shuni aytish mumkinki, STL orqali foydalanuvchi vaqtini tejashi hamda dastur tuzish jarayonida ijodiy fikrlash jarayonini yanada kengaytirishi mumkin. Men kelejakda dastur tuzganimda STL kutubxonalardan foydalanaman.

FOYDALANILGAN ADABIYOTLAR

1. Sh. M. Mirziyoyev “Tanqidiy tahlil, qat’iy tartib-intizom va shaxsiy javobgarlik – har xil rahbar faoliyatining kundalik qoidasi bo’lishi kerak. Mamlakatimizda 2016-yilda ijtimoiy-iqtisodiy rivijlantirishning asosiy yakunlari va 2017-yilga mo’ljallangan iqtisodiy dasturning eng muhim ustuvor yo’nalishlariga bag’ishlangan Vazirlar mahkamasining kengaytirilgan majlisidagi ma’ruza”. Toshkent O’zbekiston 2017.
2. Sh, M. Mirziyoyev Toshkentda bo’lib o’tgan “ijtimoiy barqarorlikni ta’minlash, muqaddas dinimizning sofligini asrash davr talabi” Toshkent 2017-yil 15-iyun.
3. Джосаттис, Николаи М. Стандартная библеотека С++: справочная руководства, 2-у изд.: Пер.с англ.- М.: ООО "И. Д. Вильямс"б 2014. – 1136 с.: ил. - Парал, тит. англ.
4. Н. Джосьютис. С++ Стандартная библеотека. Для профессионалов. - СП Питер, 2004. - 730 с.: ил
5. Вандевурд, Дэвид, Джосаттис, Николаи, М. Шаблоны С++Ж справочник разработчика: Пер. с англ. - М.: Издательский доя "Вильямс", 2003. - 544 с.: ил. - Парал, тит. англ.