O‘ZBEKISTON RESPUBLIKASI
OLIY VA O‘RTA MAXSUS TA’LIM VAZIRLIGI
O‘RTA MAXSUS, KASB-HUNAR TA’LIMI MARKAZI

SH.A.NAZIROV, R.V.QOBULOV

OBYEKTGA MO‘LJALLANGAN DASTURLASH

Kasb-hunar kollejlari uchun
o‘quv qo‘llanma

G‘afur G‘ulom nomidagi nashriyot-matbaa ijodiy uyi
Toshkent — 2007

32.973.202-018.2
N 18

Oliy va o‘rta maxsus, kasb-hunar ta’limi ilmiy-metodik birlashmalari faoliyatini muvofiqlashtiruvchi Kengash tomonidan nashrga tavsiya etilgan

Taqrizchilar: akademik **T.Bekmurodov**, professor **B.Qurmonboyev**

Nazirov Sh.A.

Obyektga mo‘ljallangan dasturlash: Kasb-hunar kollejlari uchun o‘quv qo‘ll. / Sh. A. Nazirov, R.V. Qobulov: O‘zR Oliy va o‘rta maxsus ta’lim vazirligi, O‘rta maxsus, kasb-hunar ta’limi markazi.— T.: G‘afur G‘ulom nomidagi nashriyot-matbaa ijodiy uyi, 2007. 184 b.

I. Qobulov R.V.

«Obyektga mo‘ljallangan dasturlash» o‘quv qo‘llanmasi axborot tizimlari bo‘yicha mutaxassislar tayyorlashga yaqindan yordam beradi. Dasturiy ta’minotni yaratish va uni rivojlantirish tizimli administrlash va ma’lumotlar bazalarini boshqarish tizimlarini loyihalash bilan birga kompyuter mutaxassislarining ham eng asosiy vazifalaridan biridir. Hisoblash mashinalarini ishlab chiqarish, aloqa, boshqarish tizimlari va hujjat yuritish barcha sohalarda keng qo‘llanilishi uzluksiz murakkablashib boruvchi ko‘p hajmdagi dasturiy ta’minotni talab qiladi. Yaqindagina dasturlash san’at hisoblangan bo‘lsa, hozirgi davrda mutaxassislikka, ayrim shaxslar va katta jamoalar ishiga aylandi.

Ushbu o‘quv qo‘llanmada obyektga mo‘ljallangan dasturlashning kelib chiqish omillari va rivojlanishi, asosiy tamoyillari, UML asosida loyihalash, C++ tilida obyektga yo‘naltirilgan dasturlash haqidagi ma’lumotlar keltirilgan.

O‘quv qo‘llanma akademik litsey va kollejlar o‘quvchilari va o‘qituvchilari hamda oliy o‘quv yurtlari talabalari va mustaqil o‘rganuvchilar uchun mo‘ljallangan.

ББК 32.973.202-018.2я722

N-Q $\frac{\textbf{2210010000 — 24}}{\textbf{M 352(04) — 2007}}$ **qat’iy buyurtma 2007**

ISBN 978-9943-03-041-1

1. OBYEKTGA MO‘LJALLANGAN DASTURLASH

1.1. Obyektga mo‘ljallangan yondashuv tarixi

Obyektga mo‘ljallangan yondashuv (OMY) bir kunda o‘ylab topilgan emas. Uning paydo bo‘lishi dasturiy ta’minotning tabiiy rivojidagi navbatdagi pog‘ona, xolos. Vaqt o‘tishi bilan qanday uslublar ishlash uchun qulay, qaysinisi noqulay ekanini aniqlash oson bo‘lib bordi. OMY eng muvaffaqiyatli, vaqt sinovidan o‘tgan uslublarni o‘zida mujassam etadi.

Dastlab dasturlash anchayin boshqotirma ixtiro bo‘lib, u dasturchilarga dasturlarni kommutatsiya bloki orqali kompyuterning asosiy xotirasiga to‘g‘ridan-to‘g‘ri kiritish imkonini berdi. Dasturlar mashina tillarida ikkilik tasavvurda yozilar edi. Dasturlarni mashina tilida yozishda tez-tez xatolarga yo‘l qo‘yilar, kodni kuzatib borish amalda deyarli mumkin emas edi. Bundan tashqari, mashina kodlaridagi dastur tushunish uchun g‘oyat murakkab edi.

Vaqt o‘tishi bilan kompyuterlar tobora kengroq qo‘llana boshlandi hamda yuqoriroq darajadagi protsedura tillari paydo bo‘ldi. Bularning dastlabkisi FORTRAN tili edi. Biroq obyektga mo‘ljallangan yondashuv rivojiga asosiy ta’sir keyinroq paydo bo‘lgan. Protsedura tillari dasturchiga axborotga ishlov berish dasturini pastroq darajadagi bir nechta protseduraga bo‘lib tashlash imkonini beradi. Pastroq darajadagi bunday protseduralar dasturning umumiy tuzilmasini belgilab beradi. Ushbu protseduralarga izchil murojaatlar protseduralardan tashkil topgan dasturlarning bajarilishini boshqaradi.

Dasturlashning bu yangi paradigmasi mashina tilida dasturlash paradigmasiga nisbatan ancha ilg‘or bo‘lib, unga tuzilmalashtirishning asosiy vositasi bo‘lgan protseduralar qo‘shilgan edi. Maydaroq funksiyalarni nafaqat tushunish, balki sozlash ham osonroq kechadi. Biroq, boshqa tomondan, protsedurali dasturlash koddan takroran foydalanish imkonini cheklab qo‘yadi. Buning ustiga dasturchilar tez-tez «makaron» dasturlar ham yozib turishganki, bu dasturlarni bajarish likopdagi spagetti uyumini ajratishga o‘xshab ketar edi. Va, nihoyat, shu narsa aniq bo‘ldiki, protsedurali dasturlash usullari bilan

dasturlarni ishlab chiqishda diqqatni ma'lumotlarga qaratishning o'zi muammolarni keltirib chiqarar ekan. Chunki ma'lumotlar va protsedura ajralgan, ma'lumotlar inkapsulatsiyalanmagan. Bu nimaga olib keladi? Bu har bir protsedura ma'lumotlarni nima qilish kerakligini va ular qayerda joylashganini bilmog'i lozim bo'ladi. Agar protsedura o'zini yomon tutsa-yu, ma'lumotlar ustidan noto'g'ri amallarni bajarsa, u ma'lumotlarni buzib qo'yishi mumkin. Har bir protsedura ma'lumotlarga kirish usullarini dasturlashi lozim bo'lganligi tufayli, ma'lumotlar taqdimotining o'zgarishi dasturning ushbu kirish amalga oshirilayotgan barcha o'rinlarining o'zgarishiga olib kelar edi. Shunday qilib, hatto eng kichik to'g'rilash ham butun dasturda qator o'zgarishlar sodir bo'lishiga olib kelar edi.

Modulli dasturlashda, masalan, Modula 2 kabi tilda protsedurali dasturlashda topilgan ayrim kamchiliklarni bartaraf etishga urinib ko'rildi. Modulli dasturlash dasturni bir necha tarkibiy bo'laklarga, yoki, boshqacha qilib aytganda, modullarga bo'lib tashlaydi. Agar protsedurali dasturlash ma'lumotlar va protseduralarni bo'lib tashlasa, modulli dasturlash, undan farqli o'laroq, ularni birlashtiradi. Modul ma'lumotlarning o'zidan hamda ma'lumotlarga ishlov beradigan protseduralardan iborat. Dasturning boshqa qismlariga moduldan foydalanish kerak bo'lib qolsa, ular modul interfeysiga murojaat etib qo'ya qoladi. Modullar barcha ichki axborotni dasturning boshqa qismlarida yashiradi.

Biroq modulli dasturlash ham kamchiliklardan xoli emas. Modullar kengaymas bo'ladi, bu degani kodga bevosita kirishsiz hamda uni to'g'ridan-to'g'ri o'zgartirmay turib modulni qadamma-qadam o'zgartirish mumkin emas. Bundan tashqari, bitta modulni ishlab chiqishda, uning funksiyalarini boshqasiga o'tkazmay (delegat qilmay) turib boshqasidan foydalanib bo'lmaydi. Yana garchi modulda turni belgilab bo'lsa-da, bir modul boshqasida belgilangan turdan foydalana olmaydi.

Modulli va protsedurali dasturlash tillarida tuzilmalashtirilgan va tuzilmalashtirilmagan ma'lumotlar o'z «tur»iga ega. Biroq turni kengaytirish usuli, agar «agregatlash» deb ataluvchi usul yordamida boshqa turlarni yaratishni hisobga olmaganda, mavjud emas.

Va, nihoyat, modulli dasturlash — bu yana protseduraga mo'ljallangan gibridli sxema bo'lib, unga amal qilishda dastur bir necha protseduralarga bo'linadi. Biroq endilikda protseduralar ishlov berilmagan ma'lumotlar ustida amallarni bajarmaydi, balki modullarni boshqaradi.

Obyektga mo‘ljallangan dasturlash (OMD) modulli dasturlashdan keyingi mantiqiy pog‘onani egallaydi, u modulga nasldan naslga o‘tishni va polimorfizmni qo‘shadi. Dasturchi OMD dan foydalanar ekan, dasturni bir qator oliy darajali obyektlarga bo‘lish yo‘li bilan tizimlashtiradi. Har bir obyekt hal qilinayotgan muammoning ma’-lum bir tomonini modellashtiradi. OMD endilikda dasturni bajarish jarayonini boshqarish uchun dasturchi diqqatini protseduralarni ketma-ketlikda chaqirib olish ro‘yxatini tuzib o‘tirishga qaratmaydi. Buning o‘rniga obyektlar o‘zaro aloqada bo‘ladi. OMY yordamida ishlab chiqilgan dastur hal qilinayotgan muammoning amaldagi modeli bo‘lib xizmat qiladi.

Dasturga obyektlar atamalari bilan ta’rif berish dasturiy ta’minotni ishlab chiqishning eng tushunarli usulidir. Obyektlar hamma narsani obyekt nima qilayotgani nuqtayi nazaridan idrok etishga, ya’ni uning xatti-harakatlarini xayolan modellashtirishga majbur qiladi. Shu tufayli obyektga yondashishda u dasturning bajarilishi jarayonida qanday ishlatiladi degan nuqtayi nazardan biroz e’tiborni chalg‘itishi mumkin. Shunday qilib, dasturni yozish jarayonida haqiqiy dunyoning tabiiy atamalaridan foydalanish mumkin. Dasturni alohida protseduralar va ma’lumotlar shaklida (kompyuter dunyosi atamalarida) qurish o‘rniga obyektlardan iborat dastur qurish mumkin. Obyektlar otlar, fe’llar va sifatlar yordamida haqiqiy dunyoni dasturda modellashti-rishga imkon beradi. Joriy qilish (realizatsiya) xatti-harakatlar qanday bajarilayotganini belgilaydi. Dasturlash atamalarida joriy qilish — bu dasturiy kod.

Yechilayotgan masala atamalari bilan fikrlab, joriy qilishning mayda-chuyda detallarida o‘ralashib qolish xavfidan qochish mumkin. Albatta, ayrim oliy darajadagi obyektlar kompyuter bilan aloqa qilishda past darajadagi, mashinaga mo‘ljallangan usullardan foydalanishi lozim. Biroq obyekt bu aloqani tizimning boshqa qismlaridan izolatsiya qiladi.

Obyekt dastur konstruksiyasi bo‘lib, unda holat va xatti-harakat inkapsulatsiyalangan bo‘ladi. Obyekt holati bu ichki obyekt o‘zgaruv-chanlari qiymatlarining yig‘indisidir.

Ichki o‘zgaruvchan deb obyekt ichida saqlanadigan qiymatga aytiladi.

Mohiyat-e’tibori bilan obyekt bu sinfning ekzemplyari (nusxala-ridan biri)dir.

OMD, haqiqiy dunyo kabi, obyektlardan tashkil topadi. Obyektga mo‘ljallangan sof dasturlash tilida eng dastlabki, bazaviy, butun,

mantiqiy turlardan tortib, to sinflarning murakkabroq nusxalarigacha, barchasi obyekt hisoblanadi. Biroq obyektga mo‘ljallangan tillarning hammasi ham bu darajada chuqurlashib ketmagan. Ayrim tillarda (masalan, Java kabi) int va float ga o‘xshash oddiy primitivlar obyekt sifatida olib qaralmaydi.

OMD obyektlari, haqiqiy olam obyektlari kabi, o‘z xususiyatlari va xatti-harakatlari bo‘yicha tasniflanadi.

Biologiyada itlar, mushuklar, fillar va odamlar sutemizuvchilar sinfiga kiradi. Bu turli xildagi jonivorlarni umumiy xususiyatlar birlashtirib turadi. Xuddi shuningdek, dasturiy ta’minot olamida ham obyektlar bitta yoki bir nechta sinflarga mansub bo‘ladi.

Bitta sinfga mansub obyektlarga umumiy xususiyatlar xos bo‘ladi. Boshqacha qilib aytganda, sinf obyektni tavsiflaydigan xususiyatlar va xulq-atvorlarni, shuningdek, bu eng muhimi, obyekt javob beradigan xabarlarni belgilab beradi. Biror bir obyekt boshqa obyektning xulq-atvoriga ta’sir ko‘rsatgan vaqtda u bu ta’sirni bevosita ko‘rsatmaydi, balki undan qandaydir bir qo‘shimcha axborotdan foydalangan holda o‘zini o‘zi o‘zgartirishni iltimos qiladi. Odatda bu «xabarni jo‘natish» deb ataladi.

Sinf umumiy xususiyatlar va xulq-atvorga ega bo‘lgan obyektlarni birlashtiradi. Bitta sinfga mansub obyektlar bir xil xususiyatlarga ega bo‘lib, bir xil xatti-harakat namoyon etadi.

Sinflar shablon (qolip)ga o‘xshaydi: ular obyektlarning ekzemplyarlari (nusxalari)ni tayyorlash uchun qo‘llanadi.

Belgilar — sinfning tashqaridan ko‘rinib turgan xususiyatlari.

Obyekt ichki o‘zgaruvchiga bevosita kirishni taqdim etganda yoki usul yordamida qiymatni qaytargandagina o‘z belgilarini namoyon qilishi mumkin.

Xulq-atvor — xabarga yoki holatning o‘zgarishiga javoban obyekt tomonidan bajariladigan xatti-harakatlar. U obyekt nima qilayotganini bildiradi.

Bir obyekt ikkinchi obyekt ustida xatti-harakatlar bajarib, uning xulq-atvoriga ta’sir ko‘rsatishi mumkin. «Xatti-harakat» atamasi o‘rniga «usulni chaqirish», «funksiyani chaqirish» yoki «xabarni uzatish» atamalari qo‘llanadi. Muhimi, bu atamalarning qaysi biri qo‘llanayotganida emas, albatta, muhimi, bu xatti-harakatlar obyekt xulq-atvorini namoyon qilishga da’vat etishidadir.

Obyektlar o‘rtasidagi aloqa obyektga mo‘ljallangan dasturlashning muhim tarkibiy qismidir. Obyektlar o‘zaro aloqasining ikkita asosiy usuli mavjud:

Birinchi usul: obyektlar biri ikkinchisidan mustaqil ravishda mavjud bo‘ladi. Agar alohida obyektlarga o‘zaro aloqa kerak bo‘lib qolsa, ular bir-birlariga xabar jo‘natadi.

Obyektlar bir-birlari bilan xabarlar yordamida aloqa qiladi. Xabar olgan obyekt ma’lum xatti-harakatlarni bajaradi.

Xabar uzatish bu obyekt holatini o‘zgartirish maqsadida uslubni chaqirib olish yoki xulq-atvor modellaridan birini qo‘llashning o‘zginasidir.

Ikkinchi usul: obyekt tarkibida boshqa obyektlar bo‘lishi mumkin. Xuddi OMD da bo‘lganidek, obyektlar ham, o‘z navbatida, agregatsiya yordamida boshqa obyektlardan jamlanishi mumkin. Ushbu obyektlarning har bittasida uslub va belgilarga ega bo‘lgan interfeys mavjud bo‘ladi.

Xabar — obyektga mo‘ljallangan yondashuvning muhim tushunchasi. Xabarlar mexanizmi tufayli obyektlar o‘z mustaqilligini saqlab qolishi mumkin. Boshqa biron obyektga xabar jo‘natayotgan obyekt uchun xabar olgan obyekt talabdagi xatti-harakatni qanday bajarishi unchalik muhim emas. Unga xatti-harakat bajarilganligining o‘zi muhimdir.

1.2. Obyektga mo‘ljallangan yondashuvning afzalliklari va maqsadlari

OMY dasturiy ta’minotni ishlab chiqishda oltita asosiy maqsadga asoslanadi. OMY paradigmasiga muvofiq ishlab chiqilgan dasturiy ta’minot quyidagi xususiyatlarga ega bo‘lmog‘i lozim:

1. Tabiiylik.
2. Ishonchlilik.
3. Qayta qo‘llanish imkoniyati.
4. Kuzatib borishda qulaylik.
5. Takomillashishga qodirlik.
6. Yangi versiyalarni davriy chiqarishning qulayligi.

Tabiiylik

OMY yordamida tabiiy dasturiy ta’minot yaratiladi. Tabiiy dasturlar tushunarliroq bo‘ladi. Dasturlashda «massiv» yoki «xotira sohasi» kabi atamalardan foydalanish o‘rniga, yechilayotgan masala mansub bo‘lgan soha atamalaridan foydalanish mumkin. Ishlab chiqilayotgan dasturni kompyuter tiliga moslash o‘rniga, OMY aniq bir sohaning atamalaridan foydalanish imkonini beradi.

Ishonchlilik

Yaxshi dasturiy ta'minot boshqa har qanday mahsulotlar, masalan, muzlatkich yoki televizorlar kabi ishonchli bo'lmog'i lozim.

Puxta ishlab chiqilgan va tartib bilan yozilgan obyektga mo'ljallangan dastur ishonchli bo'ladi. Obyektlarning modulli tabiati dastur qismlaridan birida, uning boshqa qismlariga tegmagan holda, o'zgartishlar amalga oshirish imkonini beradi. Obyekt tushunchasi tufayli, axborotga ushbu axborot kerak bo'lgan shaxslar egalik qiladi, mas'uliyat esa berilgan funksiyalarni bajaruvchilar zimmasiga yuklatiladi.

Qayta qo'llanish imkoniyati

Quruvchi uy qurishga kirishar ekan, har gal g'ishtlarning yangi turini ixtiro qilmaydi. Radiomuhandis yangi sxemani yaratishda har gal rezistorlarning yangi turini o'ylab topmaydi. Unda nima uchun dasturchi «g'ildirak ixtiro qilaverishi kerak?» Masala o'z yechimini topgan ekan, bu yechimdan ko'p martalab foydalanish lozim.

Malakali ishlab chiqilgan obyektga mo'ljallangan sinflarni bemalol takroran ishlatish mumkin. Xuddi modullar kabi, obyektlarni ham turli dasturlarda takroran qo'llash mumkin. Modulli dasturlashdan farqli o'laroq, OMY mavjud obyektlarni kengaytirish uchun vorislikdan, sozlanayotgan kodni yozish uchun esa polimorfizmdan foydalanish imkonini beradi.

Kuzatib borishda qulaylik

Dasturiy mahsulotning ish berish davri uning ishlab chiqilishi bilan tugamaydi. Dasturni ishlatish jarayonida *kuzatib borish* deb nomlanuvchi tirgak kerak. Dasturga sarflangan 60 foizdan 80 foizgacha vaqt kuzatib borishga ketadi. Ishlab chiqish esa ish berish siklining 20 foizinigina tashkil etadi.

Puxta ishlangan obyektga mo'ljallangan dastur ishlatish uchun qulay bo'ladi. Xatoni bartaraf etish uchun faqat bitta o'ringa to'g'rilash kiritish kifoya qiladi. Chunki ishlatishdagi o'zgarishlar tiniq, boshqa barcha obyektlar takomillashtirish afzalliklaridan avtomatik ravishda foydalana boshlaydi. O'zining tabiiyligi tufayli dastur matni boshqa ishlab chiquvchilar uchun tushunarli bo'lmog'i lozim.

Takomillashishga qodirlik

Foydalanuvchilar dasturni kuzatib borish paytida tez-tez tizimga yangi funksiyalarni qo'shishni iltimos qiladilar. Obyektlar kutubxonasini tuzishning o'zida ham ushbu obyektlarning funksiyalarini kengaytirishga to'g'ri keladi.

Dasturiy ta'minot statik (qotib qolgan) emas. Dasturiy ta'minot foydali bo'lib qolishi uchun uning imkoniyatlarini muttasil kengaytirib borish lozim. OMY da dasturni kengaytirish usullari ko'p. Vorislik, polimorfizm, qayta aniqlash, vakillik hamda ishlab chiqish jarayonida foydalanish mumkin bo'lgan ko'plab boshqa shablonlar shular jumlasidandir.

Yangi versiyalarni davriy chiqarilishning qulayligi

Zamonaviy dasturiy mahsulotning ish berish davri ko'p hollarda haftalar bilan o'lchanadi. OMY tufayli dasturlarni ishlab chiqish davrini qisqartirishga erishildi, chunki dasturlar ancha ishonchli bo'lib bormoqda, kengayishi osonroq hamda takroran qo'llanishi mumkin.

Dasturiy ta'minotning tabiiyligi murakkab tizimlarning ishlab chiqilishini osonlashtiradi. Har qanday ishlanma hafsala bilan yondashuvni talab qiladi, shuning uchun tabiiylik dasturiy ta'minotning ishlab chiqish davrlarini qisqartirish imkonini beradi, chunki butun diqqat-e'tiborni yechilayotgan masalaga jalb qildiradi.

Dastur qator obyektlarga bo'lingach, har bir alohida dastur qismini boshqalari bilan parallel ravishda hamda bir nechta ishlab chiquvchi sinflarni bir-birlaridan mustaqil ravishda ishlab chiqishi mumkin bo'ladi. Ishlab chiqishdagi bunday parallellik ishlab chiqish vaqtini qisqartiradi.

1.3. Obyektga mo'ljallangan yondashuvning uch tamoyili

Obyektga mo'ljallangan yondashuv (OMY) ni tushunib yetish hamda undan foydalanishni o'zlashtirib olish uchun, avvalambor, puxta bazaviy bilimlarni egallab olish lozim. Bazaviy tushunchalarni puxta anglab yetibgina dasturlarni yaratishda OMY ni qo'llash mumkin. Inkapsulatsiyalash, vorislik va polimorfizm obyektga mo'ljallangan dasturlash (OMD) ning uchta bazaviy tushunchasi hisoblanadi.

Inkapsulatsiyalash

Inkapsulatsiyalash dasturni qandaydir monolit, bo'linmas narsa sifatida olib qaramay, ko'plab mustaqil elementlarga bo'lish imkonini beradi. Har bir element o'z funksiyalarini boshqa elementlardan mustaqil ravishda bajara oladigan alohida modul sifatida olib qaraladi. Aynan inkapsulatsiyalash tufayli mustaqillik darajasi ortadi, chunki ichki detallar interfeys ortida yashiringan bo'ladi.

Inkapsulatsiyalash modullikning obyektga mo‘ljallangan tavsifidir. Inkapsulatsiyalash yordamida dasturiy ta’minotni ma’lum funksiyalarni bajaruvchi modullarga bo‘lib tashlash mumkin. Bu funksiyalarni amalga oshirish detallari esa tashqi olamdan yashirin holda bo‘ladi.

Mohiyatan *inkapsulatsiyalash* atamasi «germetik berkitilgan; tashqi ta’sirlardan himoyalangan dastur qismi» degan ma’noni bildiradi.

Agar biror bir dasturiy obyektga inkapsulatsiyalash qo‘llangan bo‘lsa, u holda bu obyekt qora quti sifatida olib qaraladi. Siz qora quti nima qilayotganini uning tashqi interfeysini ko‘rib turganingiz uchungina bilishingiz mumkin. Qora qutini biron narsa qilishga majburlash uchun unga xabar yuborish kerak. Qora quti ichida nima sodir bo‘layotgani ahamiyatli emas, qora quti yuborilgan xabarga adekvat (mos ravishda) munosabatda bo‘lishi muhimroqdir.

Interfeys tashqi olam bilan tuzilgan o‘ziga xos bitim bo‘lib, unda tashqi obyektlar ushbu obyektga qanday talablar yuborishi mumkinligi ko‘rsatilgan bo‘ladi. Interfeys — obyektni boshqarish pulti.

Ommaviy, xususiy va himoyalangan kirish

Qandaydir bir elementni ommaviy interfeysga kiritish yoki, aksincha, undan chiqarish uchun kalit so‘zdan foydalanish kerak. OMD ning har bir tilida kalit so‘zlar to‘plami belgilangan, biroq bu so‘zlar asosan bir xil funksiyalarni bajaradi.

Obyektga mo‘ljallangan tillarning ko‘pchiligida kirishning uchta darajasi mavjud:

1. Ommaviy (public) — barcha obyektlarga kirish uchun ruxsat bor.
2. Himoyalangan (protected) — faqat ushbu ekzemplyarga va har qanday tarmoq sinflarga kirishga ruxsat bor.
3. Xususiy (private) — faqat ushbu ekzemplyarga kirishga ruxsat bor.

Loyihada kirish darajasini to‘g‘ri tanlab olish muhim ahamiyatga ega. Ko‘rinadigan qilinishi lozim bo‘lgan barcha narsa ommaviy bo‘lmog‘i lozim. Berkitilishi lozim bo‘lgan har qanday narsa himoyalangan yoki xususiy kirishga ega bo‘lmog‘i kerak.

Inkapsulatsiyalash nima uchun kerak?

Inkapsulatsiyalashdan to‘g‘ri foydalanish tufayli obyektlar bilan o‘zgartiriladigan komponentlar (tarkibiy qismlar) dek muomala qilish mumkin. Boshqa obyekt sizning obyektingizdan foydalana olishi uchun u sizning obyektingizning ommaviy interfeysidan qanday foydalanish kerakligini bilishi kifoya. Bunday mustaqillik uchta muhim afzallikka ega:

1.Mustaqilligi tufayli obyektdan takroran foydalanish mumkin. Inkapsulatsiyalash puxta amalga oshirilgan bo‘lsa, obyektlar ma’lum bir programmaga bog‘lanib qolgan bo‘lmaydi. Ulardan imkoni bo‘lgan hamma yerda foydalanish mumkin bo‘ladi. Obyektdan boshqa biron o‘rinda foydalanish uchun uning interfeysidan foydalanib qo‘ya qolish kifoya.

2. Inkapsulatsiyalash tufayli obyektda boshqa obyektlar uchun ko‘rinmas bo‘lgan o‘zgarishlarni amalga oshirish mumkin. Agar interfeys o‘zgartirilmasa, barcha o‘zgarishlar obyektdan foydalanayotganlar uchun ko‘rinmas bo‘ladi. Inkapsulatsiyalash komponentni yaxshilash, amalga oshirish samaradorligini ta’minlash, xatolarni bartaraf etish imkonini beradi, yana bularning hammasi dasturning boshqa obyektlariga ta’sir ko‘rsatmaydi. Obyektdan foydalanuvchilar ularda amalga oshirilayotgan barcha o‘zgarishlardan avtomatik tarzda yutadilar.

3. Himoyalangan obyektdan foydalanishda obyekt va dasturning boshqa qismi o‘rtasida biror bir ko‘zda tutilmagan o‘zaro aloqalar bo‘lishi mumkin emas. Agar obyekt boshqalardan ajratilgan bo‘lsa, bu holda u dasturning boshqa qismi bilan faqat o‘z interfeysi orqali aloqaga kirishishi mumkin.

Shunday qilib, inkapsulatsiyalash yordamida modulli dasturlarni yaratish mumkin. Samarali inkapsulatsiyalashning quyidagicha uchta o‘ziga xos belgisi mavjud:

— abstraksiya;
— joriy qilishning berkitilganligi;
— mas’uliyatning bo‘linganligi.

Abstraksiya

Garchi obyektga mo‘ljallangan tillar inkapsulatsiyalashdan foydalanishga yordam bersa-da, biroq ular inkapsulatsiyalashni kafolatlamaydi. Tobe va ishonchsiz kodni yaratib qo‘yish oson. Samarali inkapsulatsiyalash — sinchkovlik bilan ishlab chiqish hamda abstraksiya va tajribadan foydalanish natijasi. Inkapsulatsiyalashdan samarali foydalanish uchun dasturni ishlab chiqishda avval abstraksiyadan va uning bilan bog‘liq konsepsiyalardan foydalanishni o‘rganib olish lozim.

Abastraksiya murakkab masalani soddalashtirish jarayonidir. Muayyan masalani yechishga kirishar ekansiz, siz barcha detallarni hisobga olishga urinmaysiz, balki yechimni osonlashtiradiganlarini tanlab olasiz.

Aytaylik, siz yo‘l harakati modelini tuzishingiz kerak. Shunisi ayonki, bu o‘rinda siz svetoforlar, mashinalar, shosselar, bir tomonlama va ikki tomonlama ko‘chalar, ob-havo sharoitlari va h.k. sinflarini yaratasiz. Ushbu elementlarning har biri transport harakatiga ta’sir ko‘rsatadi. Biroq bu o‘rinda xasharotlar va qushlar ham yo‘lda paydo bo‘lishi mumkin bo‘lsa-da, siz ularning modelini yaratmaysiz. Inchunin, siz mashinalar markalarini ham ajratib ko‘rsatmaysiz. Siz haqiqiy olamni soddalashtirasiz hamda uning faqat asosiy elementlaridan foydalanasiz. Mashina — modelning muhim detali, biroq bu Kadillakmi yoki boshqa biron markadagi mashinami, yo‘l harakati modeli uchun bu detallar ortiqcha.

Abstraksiyaning ikkita afzal jihati bor. Birinchidan, u masala yechimini soddalashtiradi. Muhimi yana shundaki, abstraksiya tufayli dasturiy ta’minot komponentlaridan takroran foydalanish mumkin. Takroran qo‘llanadigan komponentlarni yaratishda ular odatda g‘oyat ixtisoslashadi. Ya’ni komponentlar biror bir ma’lum masala yechimiga mo‘ljallangani, yana ular keraksiz o‘zaro bog‘liqlikda bo‘lgani sababli dastur fragmentining boshqa biron o‘rinda takroran qo‘llanishi qiyinlashadi. Imkoni boricha bir qator masalalarni yechishga qaratilgan obyektlarni yaratishga harakat qiling. Abstraksiya bitta masala yechimidan ushbu sohadagi boshqa masalalarni ham yechishda foydalanish imkonini beradi.

Quyidagi ikkita misolni ko‘rib chiqaylik:

Birinchi misol: bank kassiriga navbatda turgan odamlarni tasavvur qiling. Kassir bo‘shaganda, uning darchasiga navbatda turgan birinchi mijoz yaqinlashadi. Shunday qilib, navbatdagi hamma odam birin-ketin kassir darchasi tomon suriladi. Navbatda turganlar «birinchi kelganga birinchi bo‘lib xizmat ko‘rsatish» algoritmi bo‘yicha surilib boradi.

Ikkinchi misol: gazakxonada gamburgerli konveyerni ko‘rib chiqaylik. Navbatdagi yangi gamburger konveyerga kelib tushganda, u gamburgerlar qatoridagi oxirgi gamburger yonidan joy oladi. Shuning uchun konveyerdan olingan gamburger u yerda boshqalaridan ko‘proq vaqt turib qolgan bo‘ladi. Restoranlar «birinchi kelganga birinchi bo‘lib xizmat ko‘rsatish» algoritmi bo‘yicha ishlaydi.

Garchi bu misollar butkul turlicha bo‘lsa-da, ularda qandaydir umumiy tamoyil qo‘llangan bo‘lib, undan boshqa vaziyatlarda ham foydalanish mumkin. Boshqacha qilib aytganda, siz abstraksiyaga kelasiz.

Bu misollarning har ikkalasida ham «birinchi kelganga birinchi bo‘lib xizmat ko‘rsatish» algoritmi qo‘llangan. Bu o‘rinda navbat

elementi nimani bildirishi muhim emas. Haqiqatda ushbu element navbat oxiriga kelib qo‘shilishi hamda navbatni uning boshiga yetganda tark etishigina muhimdir.

Abstraksiya yordamida bir marta navbatni yaratib, keyinchalik uni boshqa dasturlarni yozishda qo‘llash mumkinki, bu dasturlarda elementlarga «birinchi kelganga birinchi bo‘lib xizmat ko‘rsatish» algoritmi bo‘yicha ishlov beriladi.

Samarali abstraksiyani bajarish uchun bir nechta qoidalarni ifodalash mumkin:

— Qandaydir aniq holatni emas, umumiy holatni olib qarang.

— Turli masalalarga xos bo‘lgan umumiy jihatni izlab toping. Shunchaki alohida hodisani emas, asosiy tamoyilni ko‘ra bilishga harakat qiling.

— Garchi abstraksiya g‘oyat qimmatli bo‘lsa-da, biroq eng yechimli masalani yodingizdan chiqarmang.

— Abstraksiya hammavaqt ham ochiq-oydin emas. Masalani yechar ekansiz, siz birinchi, ikkinchi va, hatto, uchinchi marta ham abstraksiyani tanib ololmasligingiz mumkin.

— Muvaffaqiyatsizlikka tayyor turing. Amalda har bir vaziyat uchun to‘g‘ri keladigan abstrakt dasturni yozish mumkin emas.

Abstraksiyani so‘nggi maqsad sifatida emas, balki unga erishish yo‘lidagi vosita sifatida olib qarash kerak. Muayyan hollarda abstraksiyani qo‘llash kerak emas. Agar evristik qoida mavjud bo‘lib, unga ko‘ra, siz biror bir masalani o‘zaro o‘xshash usullar bilan kamida uch marta yechgan bo‘lsangiz, abstraksiyani faqat shunday masalalarga qo‘llash tavsiya qilinadi.

Abstrakt komponentni takroran qo‘llash osonroq, chunki u biror bir bitta o‘ziga xos masalani yechishga emas, balki qator masalalarni yechishga mo‘ljallangan. Biroq bu hol komponentdan shunchaki takroran foydalanishdan ko‘ra ko‘proq inkapsulatsiyalashga tegishli. Ichki detallarni yashirishga o‘rganish g‘oyat muhimdir. Ma’lumotlarning abstrakt turlarini qo‘llash inkapsulatsiyalashni samarali qo‘llashga imkon beradi.

Joriy qilishni yashirish yordamida sirlarni yashirish

Abstraksiya samarali inkapsulatsiyalashning tarkibiy qismlaridan biri, xolos. Tashqi ta’sirlardan mutlaqo himoyalanmagan abstrakt dasturni ham yozish mumkin. Aynan shuning uchun obyektning ichki joriy qilinishini berkitish kerak bo‘ladi.

Joriy qilishning berkitilganligi

Joriy qilishning berkitilganligi ikkita afzallikka ega:

— obyektlarni foydalanuvchilardan himoyalaydi;

— foydalanuvchilarni obyektlardan himoyalaydi.

Birinchi afzallik — obyektlarni himoyalashni ko‘rib chiqamiz.

Asl inkapsulatsiyalash til darajasida qurilma til konstruksiyalari yordamida ta’minlanadi.

Ma’lumotlarning abstrakt turlari — bu ma’lumotlar va ular ustida o‘tkaziladigan operatsiyalar to‘plami.

Ma’lumotlarning abstrakt turlari ichki axborot va holatni puxta ishlab chiqilgan interfeys ortida yashirar ekan, ular tilda ma’lumotlarning yangi turlarini aniqlashga imkon beradi. Bunday interfeysda ma’lumotlarning abstrakt turlari bo‘linmas butunlik sifatida taqdim etilgan. Ma’lumotlarning abstrakt turlari inkapsulatsiyalashni qo‘llashni osonlashtiradi, chunki ular tufayli inkapsulatsiyalashni vorisliksiz va polimorfizmsiz qo‘llash mumkin, bu esa inkapsulatsiyalashning aynan o‘ziga diqqatni qaratish imkonini beradi. Ma’lumotlarning abstrakt turlari, shuningdek, tur tushunchasining qo‘llanishini ham osonlashtiradi. Agar tur nima ekanini anglab olsak, bu holda obyektga mo‘ljallangan yondashuv ixtisoslashtirilgan foydalanuvchilik turlari yordamida tilni kengaytirishning tabiiy usulini taklif qilayotganini oson sezib olish mumkin.

Dasturlashda qator o‘zgaruvchilar yaratiladi va ularga qiymatlar beriladi. Turlar yordamida dastur uchun qulay bo‘lgan turli ko‘rinishdagi qiymatlar aniqlanadi. Shunday qilib, turlar dastur komponentlaridan biri deb aytish mumkin bo‘ladi. Oddiy turlarga misol sifatida butun, uzun va suzuvchi turlarni keltirish mumkin. O‘zgaruvchining turi ushbu o‘zgaruvchi qanday qiymatlarni olishi va uning ustida qanday operatsiyalarni bajarish mumkinligini belgilab beradi.

Turlar dasturda qo‘llash mumkin bo‘lgan o‘zgaruvchilar turini aniqlab beradi. Ushbu turdagi o‘zgaruvchi qanday yo‘l qo‘yiladigan qiymatlarga ega bo‘lishi mumkinligini tur belgilab beradi. Tur nafaqat yo‘l qo‘yiladigan qiymatlar sohasini, balki ushbu o‘zgaruvchi ustida qanday operatsiyalarni bajarish mumkinligi, shuningdek, olinadigan natijalar qanday turda bo‘lishligini ham belgilab beradi.

Turlar — hisoblarda bir butunlik sifatida amal qiladigan narsa. Masalan, butun sonni olaylik. Ikkita butun sonni qo‘shar ekansiz, garchi bu sonlar kompyuter xotirasida bitlar ko‘rinishida namoyon bo‘lsa-da, siz bitlar ustidagi operatsiyalar haqida bosh qotirib o‘tirmaysiz.

Joriy etish berkitilgan bo‘lgani tufayli, obyekt ko‘zda tutilmagan va destruktiv (tuzilmani buzadigan) foydalanishdan himoyalangan bo‘ladi. Bu joriy qilish berkitilganligining afzalliklaridan biridir. Biroq joriy qilishning berkitilganligi obyektlardan foydalanuvchilar uchun ham muhim.

Joriy qilishning berkitilganligi dasturni moslashuvchan qiladi, chunki foydalanuvchilar obyektning joriy qilinishini hisobga olishga majbur emaslar. Shunday qilib, joriy qilishning berkitilganligi nafaqat obyektni himoyalaydi, balki kuchsiz bog‘langan kodni yaratishga yordam berib, ushbu obyektdan foydalanuvchilar uchun muayyan noqulayliklarni chetlab o‘tish imkonini beradi.

Kuchsiz bog‘langan kod — bu boshqa komponentlarning joriy qilinishiga bog‘liq bo‘lmagan kod.

Kuchli bog‘langan kod yoki bevosita aloqalarga ega kod — bu boshqa komponentlarning joriy qilinishi bilan uzviy bog‘liq bo‘lgan kod.

Inkapsulatsiyalash va joriy qilinishning berkitilganligi − mo‘jiza emas. Interfeys o‘zgartirilganda, eski interfeysga bog‘liq bo‘lgan eski kodni ham o‘zgartirish kerak bo‘ladi. Agar dasturni yozishda detallar interfeysda berkitilgan bo‘lsa, buning natijasida kuchsiz bog‘langan dastur yuzaga keladi.

Kuchli bog‘langan dasturda inkapsulatsiyalashning afzalliklari yo‘qoladi: mustaqil va takroran qo‘llanadigan obyektlarning yaratilishi mumkin bo‘lmaydi.

Joriy qilishning berkitilganligi o‘z kamchiliklariga ham ega. Ba’zida interfeys yordamida olish mumkin bo‘lganidan ko‘proq axborot kerak bo‘lib qoladi. Dasturlar olamida ma’lum aniqlik bilan, ya’ni ma’lum bir to‘g‘ri keladigan razryadlar miqdori bilan ishlaydigan qora qutilar kerak. Masalan, shunday vaziyat yuz berishi mumkinki, sizga 64 bitli butun sonlar kerak bo‘lib qoladi, chunki siz juda katta sonlar ustida amallar bajaryapsiz. Interfeysni belgilashda uni taqdim etishgina emas, balki joriy qilishda qo‘llangan turlarning o‘ziga xos tomonlarini hujjatlashtirish ham g‘oyat muhimdir. Biroq, ommaviy interfeysning har qanday boshqa qismi kabi, xulq-atvorni belgilagandan so‘ng uni o‘zgartirib bo‘lmaydi.

Joriy qilishni berkitib, mustaqil, boshqa komponentlar bilan kuchsiz bog‘langan dasturni yozish mumkin. Kuchsiz bog‘langan dastur mustahkamroq bo‘ladi, bundan tashqari uni modifikatsiya qilish ham osonroq. Bular tufayli esa uni takroran qo‘llash va takomillashtirish oson kechadi, chunki tizimning bitta qismidagi o‘zgarishlar uning boshqa mustaqil qismlariga ta’sir qilmaydi.

Mas'uliyatning bo'linganligi

Joriy qilishning berkitilganligi mas'uliyat tushunchasi bilan bog'liqligi tabiiydir. Kuchsiz bog'langan dasturni yaratish uchun mas'uliyatni tegishli ravishda taqsimlash ham muhimdir. Mas'uliyat tegishli ravishda taqsimlanganda, har bir obyekt o'zi mas'ul bo'lgan bitta funksiyani bajaradi hamda bu funksiyani yaxshi bajaradi. Bu esa obyekt bir butunlikni tashkil etishini ham bildiradi. Boshqacha qilib aytganda, funksiyalar va o'zgaruvchilarning tasodifiy to'plamiga ehtiyoj bo'lmaydi. Inkapsulatsiyalanayotgan obyektlar o'rtasida yaqin konseptual aloqa bo'lmog'i, barcha funksiyalar umumiy vazifani bajarmog'i kerak.

Joriy qilish berkitilmas ekan, mas'uliyat obyektdan chetga chiqib ketishi mumkin. Biroq o'z vazifasini qanday hal qilishni aynan obyektning o'zi bilishi lozim, ya'ni aynan obyekt o'z vazifasini bajarish algoritmiga ega bo'lishi kerak. Agar joriy qilish ochiq qoldirilsa, foydalanuvchi undan to'g'ridan-to'g'ri foydalanishi va shuning bilan mas'uliyatni bo'lishi mumkin.

Agar ikkita obyekt bir xil vazifani bajarsa, demak, mas'uliyat tegishlicha taqsimlanmagan bo'ladi. Dasturda ortiqcha mantiqiy sxemalar mavjud bo'lsa, uni qayta ishlash lozim bo'ladi.

Hayotda bo'lganidek, bilimlar va mas'uliyat ishni qanday qilib yaxshi bajarish mumkinligini bilgan kishiga vakolat qilinishi kerak. Bitta obyektga bitta (har holda kam miqdordagi) vazifa uchun mas'uliyatni yuklash lozim. Agar bitta obyektga ko'p miqdordagi vazifalar ustidan mas'uliyat yuklatib qo'yilgan bo'lsa, ularni bajarish murakkablashadi, obyektni kuzatib borish va takomillashtirish ham qiyinlashadi. Mas'uliyatni o'zgartirish ham xavfli, chunki bunda, agar obyekt bir necha xulq-atvor liniyalariga ega bo'lsa, ularni ham o'zgartirishga to'g'ri keladi. Natijada juda katta miqdordagi axborot bir yerga jamlanib qoladi, uni esa teng taqsimlash lozim. Obyekt juda kattalashib ketgan hollarda, u amalda mustaqil dasturga aylanadi hamda protsedurali dasturlash afzalliklaridan foydalanish bilan birga uning barcha tuzoqlariga ham ilinib qolishi mumkin bo'ladi. Natijada siz inkapsulatsiyalash umuman qo'llanmagan dasturda yuzaga keladigan barcha muammolarga duch kelib qolasiz.

Obyekt bir-ikkitadan ortiq vazifa uchun mas'ul ekanini aniqlagach, mas'uliyatning bir qismini boshqa obyektga olib o'tish kerak.

Joriy etishning berkitilganligi — samarali inkapsulatsiyalash yo'lidagi qadamlardan biri, xolos. Mas'uliyatni tegishli ravishda taqsimlamasdan siz protseduralar ro'yxatiga ega bo'lib qolasiz, xolos.

Samarali inkapsulatsiyalash=abstraksiya+joriy qilishning berkitilganligi+mas'uliyat.

Abstraksiyani olib tashlab, dasturdan takroran foydalanib bo'lmaydi. Joriy qilishning berkitilganligini olib tashlab siz kuchli bog'langan dasturga ega bo'lasiz. Nihoyat, mas'uliyatni olib tashlash natijasida esa siz protsedurali, ma'lumotlar ishloviga mo'ljallangan, markazlashmagan kuchli bog'langan dasturga ega bo'lasiz.

Inkapsulatsiyalash: namunaviy xatolar

Abstraksiyani o'ta darajada qo'llash sinfni yozishda ma'lum muammolarni keltirib chiqarishi mumkin. Barcha foydalanuvchilarga hamda barcha vaziyatlarda birdek to'g'ri keladigan sinfni yozish mumkin emas.

Haddan tashqari abstaksiyalash ham xavfli bo'lishi mumkin. Hatto agar siz biror bir elementning ishlab chiqilishida abstraksiyadan foydalangan bo'lsangiz, u shu bir elementda ham barcha vaziyatlarda ishlay olmasligi mumkin. Foydalanuvchining barcha ehtiyojlarini qondira oladigan sinfni yaratish juda qiyin. Abstraksiyaga o'ralashib qolish kerak emas, birinchi galda qo'yilgan masalani yechish kerak.

Sinfda masalani yechish uchun kerak bo'lganidan ko'proq narsani kiritish tavsiya qilinmaydi. Birdaniga barcha masalalarni yechmang, e'tiboringizni bittasining yechimiga qarating. Va shundan so'nggina qilib bo'lingan ishga nisbatan abstraksiyani qo'llash usulini izlab ko'rish mumkin.

Masalan, bahaybat hisoblar yoki murakkab modelga o'xshash ancha murakkab masalalar ham uchraydi. Bu o'rinda gap mas'uliyatni taqsimlash nuqtayi nazaridan murakkablik haqida bormoqda. Obyektning mas'uliyat sohalari qancha ko'p bo'lsa, u shuncha murakkabroq bo'ladi va uni qo'llab-quvvatlash ham ancha murakkablik tug'diradi.

Va, nihoyat, dasturlashda abstraksiyalashdan foydalanishga o'rganish uchun vaqt kerak. Haqiqiy abstrakt dastur haqiqiy hayot talablariga asoslangan bo'lmog'i lozim. U dasturchi shunchaki takroran qo'llanadigan obyektni yaratishga jazm qilganligi natijasida yuzaga kelmaydi. Aytganlaridek, ixtiroga ehtiyoj tug'ilganidagina, u tug'iladi. Xuddi shu tamoyil obyektlarni yaratishda ham amal qiladi. Birinchi martadayoq haqiqatan abstrakt, takroran qo'llanadigan obyektni yozish mumkin emas. Odatda takroran qo'llanadigan obyektlar ishda sinovdan o'tib bo'lgan hamda ko'plab o'zgarishlarga uchragan dasturni takomillashtirish jarayonida yaratiladi.

Ichki o'zgaruvchilarni hammavaqt berkitish kerak: ular konstantalar bo'lgan holatlar bundan mustasno. Muhimi, ular nafaqat berki-

tilgan bo‘lishi lozim, balki ularga faqat sinfning o‘zi kirish huquqiga ega bo‘lishi kerak. Ichki o‘zgaruvchilarga kirishga ruxsat berilganda, joriy qilish ochiladi.

Ichki ma’lumotlari boshqa nom ostida tashqi foydalanish uchun taqdim etilgan interfeysni yaratishga ehtiyoj yo‘q. Interfeys oliy darajadagi xulq-atvor yo‘llariga ega bo‘lishi lozim.

Inkapsulatsiyalashning asosiy afzalliklari

Inkapsulatsiyalash yordamida mas’uliyatni inson nuqtayi nazaridan tabiiy ko‘ringan usul bilan taqsimlash mumkin. Abstraksiyadan foydalanib, masala yechimini joriy qilish atamalarida emas, balki ushbu yechilayotgan masala mansub bo‘lgan soha atamalarida ifodalash mumkin. Abstraksiya masaladagi muhim jihatni ajratib ko‘rsatish imkonini beradi.

Kodning muhim qismlarini to‘sib va joriy qilinishni berkitib har bir alohida komponentning to‘g‘riligini tekshirib ko‘rish mumkin. Tekshirilgan komponent qo‘llanganda, har bir modulni sinchiklab tekshirish imkoni tug‘iladi, bu esa butun dasturning ishonchli ekaniga shubha qoldirmaydi. Shunday bo‘lsa-da, dastur to‘g‘ri ishlayotganiga amin bo‘lish uchun umumiy tekshiruv zarur.

Takroran qo‘llash imkoniyati: abstraksiya yordamida turli vaziyatlarda qo‘llash uchun yaroqli bo‘lgan oson o‘zgartiriladigan dasturni yaratish mumkin.

Kuzatib borishdagi qulaylik: himoyalangan dasturni kuzatib borish oson. Tobe kodni o‘zgartirmay turib sinfning joriy qilinishiga har qanday kerakli o‘zgarishlarni kiritish mumkin. Bu o‘zgarishlar joriy qilinishdagi o‘zgarishlarni ham, interfeysga yangi usullarni qo‘shishni ham o‘z ichiga olishi mumkin. Faqat interfeys semantikasi (mazmuni)ning o‘zgarishlari tobe koddagi o‘zgarishlarni talab qiladi.

Takomillashtirish: dasturni buzmay turib joriy qilinishni o‘zgartirish mumkin. Boshqacha qilib aytganda, mavjud kodning ishga layoqatliligini saqlagan holda funksional tavsiflarni takomillashtirish mumkin. Buning ustiga, joriy qilish berkitilgan ekan, takomillashtirilgan komponentdan foydalanayotgan kodning ishga tushirilish tavsiflari avtomatik tarzda yaxshilanadi: axir kod, garchi u o‘zgarmagan bo‘lsa-da, takomillashtirilgan komponentlardan foydalanadi-ku! Biroq o‘zgartishlar kiritilganidan so‘ng, yana modulni tekshirish kerak bo‘ladi. Obyektning o‘zgarishi ushbu obyekt foydalanayotgan butun kodda domino effektini keltirib chiqarishi mumkin.

Yangi versiyalarni davriy chiqarish (nashr etish) qulayligi: dasturni mustaqil modullarga bo‘lib, kodni ishlab chiqish bilan bog‘liq

vazifani bir nechta ishlab chiquvchilar o‘rtasida taqsimlash hamda shu yo‘l bilan ishlab chiqish jarayonini tezlashtirishga erishish mumkin.

Komponentlarni ishlab va tekshirib chiqib, ularni yangidan qaytadan o‘zgartirish kerak bo‘lmaydi. Shunday qilib, dasturchi bu komponentlarni takroran qo‘llashi hamda ularni yana «nol»dan boshlab yaratish uchun vaqt sarflamasligi mumkin.

Vorislik

Vorislik mavjud bo‘lgan sinfning ta’rifi asosidayoq yangi sinfni yaratish imkonini beradi. Yangi sinf boshqasi asosida yaratilgach, uning ta’rifi avtomatik tarzda mavjud sinfning barcha xususiyatlari, xulq-atvori va joriy qilinishiga vorislik qiladi. Avval mavjud bo‘lgan sinf interfeysining barcha metodlari va xususiyatlari avtomatik tarzda voris interfeysida paydo bo‘ladi. Vorislik voris sinfida biror bir jihatdan to‘g‘ri kelmagan xulq-atvorni avvaldan ko‘ra bilish imkonini beradi. Bunday foydali xususiyat dasturiy ta’minotni talablarning o‘zgarishiga moslashtirish imkonini beradi. Agar o‘zgartirishlar kiritishga ehtiyoj tug‘ilsa, bu holda eski sinf funksiyalariga vorislik qiluvchi yangi sinf yozib qo‘ya qolinadi. Keyin o‘zgartirilishi lozim bo‘lgan funksiyalarga qaytadan ta’rif beriladi hamda yangi funksiyalar qo‘shiladi. Bunday o‘rniga o‘rin qo‘yishning mazmuni shundan iboratki, u dastlabki sinf ta’rifini o‘zgartirmay turib, obyekt ishini o‘zgartirish imkonini beradi. Axir bu holda qayta test sinovlaridan puxta o‘tkazilgan asosiy sinflarga tegmasa ham bo‘ladi-da.

Agar siz ko‘p martalab qo‘llash yoki boshqa biron maqsadlarga ko‘ra vorislikni qo‘llashga ahd qilsangiz, avval har gal qarang — merosxo‘r—sinf bilan vorislikni berayotgan sinfning turlari o‘zaro mos keladimi? Vorislikda turlarning mos kelishi ko‘pincha «Is-a» testi deb ataladi. Ikkita sinf bir xil turga ega bo‘lgandagina o‘zaro «Is-a» munosabatida turibdi deb hisoblanadi.

Birinchi sinf o‘zida ikkinchi sinfning ekzemplyariga ega bo‘lgandagina ikkita sinf o‘zaro «Has-a» munosabatida turibdi deb hisoblanadi.

Aytaylik, Canine (Itlarniki) bazaviy sinfi mavjud. U holda it itniki bo‘ladi (A dog is a canine). Shuning uchun Dog sinfi Canine sinfining hosilasi bo‘lmog‘i kerak. Shuning bilan birga itning dumi (Tail) bor (A dog has a tail). Shu sababli Tail sinfining ekzemplyari (nusxasi)ni, masalan, Canine ga kiritib qo‘yish kerak. Ushbu biroz biologik misoldan ko‘rinib turganidek, «Is-a» munosabati sinflarning tabaqalanishida namoyon bo‘ladi. Sinflar orasidagi «Has-a» munosabati esa bir sinf ikkinchisida mavjud bo‘lishiga olib keladi.

Boshqa sinfga merosxo‘r bo‘layotgan sinf meros berayotgan sinf bilan shunday munosabatda bo‘lmog‘i lozimki, bunda natijaviy munosabatlar o‘z ma’nosiga ega bo‘lmog‘i, ya’ni vorislik tabaqalanishiga amal qilinishi kerak.

Vorislik — sinflar o‘rtasida «Is-a» munosabatlarini o‘rnatish imkonini beradigan mexanizm. Merosxo‘r sinf o‘z ajdodi bo‘lgan sinfdan xususiyatlar va xulq-atvorni meros qilib olayotganida, u shuningdek, o‘z ajdodi bo‘lgan sinf, ehtimol, boshqa sinflardan meros qilib olgan xususiyatlar va xulq-atvorga ham ega bo‘ladi.

Vorislik tabaqalanishi qandaydir ma’no kasb etishi uchun ajdodlar ustidan qanday amallar bajarilgan bo‘lsa, avlodlar ustidan ham shunday amallar bajarilish imkoniyati bo‘lishi lozim. Bu «Is-a» testi yordamida tekshiriladi. Merosxo‘r sinfga funksiyalarni kengaytirish va yangilarini qo‘shish uchun ruxsat beriladi. Ammo unga funksiyalarni chiqarib tashlashga ruxsat yo‘q.

Vorislik yordamida qurilgan sinf metodlar va xususiyatlarning uchta ko‘rinishiga ega bo‘lishi mumkin:

— O‘rniga qo‘yish (almashtirish): yangi sinf ajdodlarining metodi yoki xususiyatini shunchaki o‘zlashtirib olmaydi, balki unga yangi ta’rif ham beradi.

— Yangisini qo‘shish: yangi sinf butunlay yangi metodlar yoki xususiyatlarni qo‘shadi.

— Rekursiv: yangi sinf o‘z ajdodlari metodlari yoki xususiyatlarini to‘g‘ridan-to‘g‘ri olib qo‘ya qoladi.

Obyektga mo‘ljallangan tillarning ko‘pchiligi ta’rifni ma’lumot uzatilgan obyektdan qidiradilar. Agar u yerdan ta’rif topishning iloji bo‘lmasa, biron ta’rif topilmaguncha qidiruv tabaqalar bo‘yicha yuqoriga ko‘tarilaveradi. Ma’lumotni boshqarish aynan shunday amalga oshiriladi hamda aynan shu tufayli o‘ringa o‘rin qo‘yish jarayoni ish bajaradi.

Voris sinflar himoyalangan kirish darajasiga ega bo‘lgan metodlar va xususiyatlarga kirish huquqini olishlari mumkin. Bazaviy sinfda faqat avlodlar foydalanishi mumkinligi aniq bo‘lgan metodlargagina himoyalangan kirish darajasini bering. Boshqa hollarda xususiy yoki ommaviy kirish darajasidan foydalanish lozim. Bunday yondashuv barcha sinflarga, shu jumladan, tarmoq sinflarga ham kirish huquqi berilganidan ko‘ra, mustahkamroq konstruksiyani yaratish imkonini beradi.

Vorislik turlari

Vorislik uch asosiy hollarda qo‘llanadi:

1. Ko‘p martalab foydalanishda.

2. Ajralib turish uchun.

3. Turlarni almashtirish uchun.

Vorislikning ayrim turlaridan foydalanish boshqalaridan ko‘ra afzalroq hisoblanadi. Vorislik yangi sinfga eski sinfning amalda qo‘llanishidan ko‘p martalab foydalanish imkonini beradi. Kodni qirqib tashlash yoki kiritish o‘rniga, vorislik kodga avtomatik tarzda kirishni ta’minlaydi, ya’ni kodga kirishda u yangi sinfning bir qismidek olib qaraladi. Ko‘p martalab qo‘llash uchun vorislikdan foydalanar ekansiz, siz meros qilib olingan realizatsiya (joriy qilinish) bilan bog‘liq bo‘lasiz. Vorislikning bu turini ehtiyotkorlik bilan qo‘llash lozim. Yaxshisi bu o‘rinda «Has-a» munosabatidan foydalanish kerak.

Farqlash uchun vorislik faqat avlod-sinf va ajdod-sinf o‘rtasidagi farqlarni dasturlash imkonini beradi. Farqlarni dasturlash g‘oyat qudratli vositadir. Kodlash hajmining kichikligi va kodning oson boshqarilishi loyiha ishlanmasini osonlashtiradi. Bu holda kod satrlarini kamroq yozishga to‘g‘ri keladiki, bu qo‘shiladigan xatolar miqdorini ham kamaytiradi.

Almashtirish imkoniyati — OMY da muhim tushunchalardan biri. Merosxo‘r sinfga uning ajdodi bo‘lmish sinfga yuboriladigan xabarlarni yuborish mumkin bo‘lgani uchun ularning har ikkalasiga bir xil munosabatda bo‘lish mumkin. Aynan shuning uchun merosxo‘r sinfni yaratishda xulq-atvorni chiqarib tashlash mumkin emas. Almashtirish imkoniyatini qo‘llab, dasturga har qanday tarmoq turlarni qo‘shish mumkin. Agar dasturda ajdod qo‘llangan bo‘lsa, bu holda u yangi obyektlardan qanday foydalanishni biladi.

Polimorfizm

Agar inkapsulatsiyalash va vorislikni OMY ning foydali vositalari sifatida olib qarash mumkin bo‘lsa, polimorfizm — eng universal va radikal vositadir. Polimorfizm inkapsulatsiyalash va vorislik bilan chambarchas bog‘liq, boz ustiga, polimorfizmsiz OMY samarali bo‘lolmaydi. Polimorfizm — OMY paradigmasida markaziy tushunchadir. Polimorfizmni egallamay turib OMY dan samarali foydalanish mumkin emas.

Polimorfizm shunday holatki, bunda qandaydir bitta narsa ko‘p shakllarga ega bo‘ladi. Dasturlash tilida «ko‘p shakllar» deyilganda, bitta nom avtomatik mexanizm tomonidan tanlab olingan turli kodlarning nomidan ish ko‘rishi tushuniladi. Shunday qilib, polimorfizm yordamida bitta nom turli xulq-atvorni bildirishi mumkin.

Vorislik polimorfizmning ayrim turlaridan foydalanish uchun zarur. Aynan o‘rindoshlik imkoniyati mavjud bo‘lgani uchun polimor-

fizmdan foydalanish mumkin bo‘ladi. Polimorfizm yordamida tizimga to‘g‘ri kelgan paytda qo‘shimcha funksiyalarni qo‘shish mumkin. Dasturni yozish paytida hatto taxmin qilinmagan funksionallik bilan yangi sinflarni qo‘shish mumkin, buning ustiga bularning hammasini dastlabki dasturni o‘zgartirmay turib ham amalga oshirish mumkin. Yangi talablarga osongina moslasha oladigan dasturiy vosita deganda mana shular tushuniladi.

Polimorfizmning uchta asosiy turi mavjud:

— Qo‘shilish polimorfizmi.

— Parametrik polimorfizm.

— Ortiqcha yuklanish.

Qo‘shilish polimorfizmini ba’zida sof polimorfizm deb ham ataydilar. Qo‘shilish polimorfizmi shuning bilan qiziqarliki, u tufayli tarmoq sinf nusxalari o‘zini turlicha tutishi mumkin. Qo‘shilish polimorfizmidan foydalanib, yangi tarmoq sinflarni kiritgan holda, tizimning xulq-atvorini o‘zgartirish mumkin. Uning bosh afzalligi shundaki, dastlabki dasturni o‘zgartirmay turib yangi xulq-atvorni yaratish mumkin.

Aynan polimorfizm tufayli joriy qilishdan takroran foydalanishni vorislik bilan almashtirish kerak emas. Buning o‘rniga vorislikdan avvalambor o‘zaro almashinish munosabatlari yordamida polimorf xulq-atvorga erishish uchun foydalanish lozim. Agar o‘zaro almashinish munosabatlari to‘g‘ri belgilansa, buning ortidan albatta takroran qo‘llash chiqib keladi. Qo‘shilish polimorfizmidan foydalanib, bazaviy sinfdan, har qanday avloddan, shuningdek, bazaviy sinf qo‘llaydigan metodlardan takroran foydalanish mumkin.

Parametrik polimorfizmdan foydalanib turdosh metodlar va turdosh (universal) turlar yaratish mumkin. Turdosh metodlar va turlar dalillarning ko‘plab turlari bilan ishlay oladigan dasturni yozish imkonini beradi. Agar qo‘shilish polimorfizmidan foydalanish obyektni idrok etishga ta’sir ko‘rsatsa, parametrik polimorfizmdan foydalanish qo‘llanayotgan metodlarga ta’sir ko‘rsatadi. Parametrik polimorfizm yordamida parametr turini bajarilish vaqtigacha e’lon qilmay turib turdosh metodlar yaratish mumkin. Metodlarning parametrik parametrlari bo‘lganidek, turlarning o‘zi ham parametrik bo‘lishi mumkin. Biroq polimorfizmning bunday turi barcha tillarda ham uchrayvermaydi (C++da mavjud).

Ortiqcha yuklanish yordamida bitta nom turlicha metodlarni bildirishi mumkin. Bunda metodlar faqat miqdorlari va parametr turlari bilan farqlanadi. Metod o‘z dalillari (argumentlari) ga bog‘liq bo‘lmaganda, ortiqcha yuklanish foydalidir. Metod o‘ziga xos

parametrlar turlari bilan cheklanmaydi, balki har xil turdagi parametrlarga nisbatan ham qo‘llanadi. Masalan, max metodini ko‘rib chiqaylik. Maksimal — turdosh tushuncha bo‘lib, u ikkita muayyan parametrlarni qabul qilib, ularning qaysi biri kattaroq ekanini ma’lum qiladi. Ta’rif butun sonlar yoki suzuvchi nuqtali sonlar qiyoslanishiga qarab o‘zgarmaydi.

Polimorfizmdan samarali foydalanish sari qo‘yilgan birinchi qadam bu inkapsulatsiyalash va vorislikdan samarali foydalanishdir. Inkapsulatsiyalashsiz dastur osongina sinflarning joriy qilinishiga bog‘liq bo‘lib qolishi mumkin. Agar dastur sinflarning joriy qilinish aspektrlaridan biriga bog‘liq bo‘lib qolsa, tarmoq sinfda bu joriyni to‘g‘rilash mumkin bo‘lmaydi.

Vorislik — qo‘shilish polimorfizmining muhim tarkibiy qismi. Hammavaqt bazaviy sinfga imkon darajada yaqinlashtirilgan darajada dasturlashga uringan holda o‘rinbosarlik munosabatlarini o‘rnatishga harakat qilish kerak. Bunday usul dasturda ishlov berilayotgan obyektlar turlari miqdorini oshiradi.

Puxta o‘ylab ishlab chiqilgan tabaqalanish o‘rinbosarlik munosabatlarini o‘rnatishga yordam beradi. Umumiy qismlarni abstrakt sinflarga olib chiqish kerak hamda obyektlarni shunday dasturlash kerakki, bunda obyektlarning ixtisoslashtirilgan nusxalari emas, balki ularning o‘zlari dasturlashtirilsin. Bu keyinchalik har qanday voris sinfni dasturda qo‘llash imkonini beradi.

Agar til vositalari bilan interfeys va joriy qilinishni to‘liq ajratish mumkin bo‘lsa, u holda odatda mana shu vositalardan foydalanish kerak, vorislikdan emas. Interfeys va joriy qilinishni aniq ajratib, o‘rinbosarlik imkoniyatlarini oshirish va shuning bilan polimorfizmdan foydalanishning yangi imkoniyatlarini ochib berish mumkin.

Biroq ko‘p o‘rinlarda tajribasiz loyihachilar polimorfizmni kuchaytirish maqsadida xulq-atvorni juda baland tabaqaviy darajaga olib chiqishga urinadilar. Bu holda har qanday avlod ham bu xulq-atvorni ushlab tura oladi. Shuni esdan chiqarmaslik kerakki, avlodlar o‘z ajdodlarining funksiyalarini chiqarib tashlay olmaydilar. Dasturni yanada polimorf qilish maqsadida puxta rejalashtirilgan vorislik tabaqalarini buzish yaramaydi.

Hamma narsaning hisob-kitobi bor. Haqiqiy polimorfizmning kamchiligi shundaki, u unumdorlikni pasaytiradi. Polimorfizmdan foydalanganda dasturni bajarish paytida tekshiruvlar o‘tkazish talab qilinadi. Bu tekshiruvlar turlari statik ravishda berilgan qiymatlarga ishlov berishga qaraganda ko‘proq vaqt talab qiladi.

1.4. Obyektga mo‘ljallangan tahlil va loyihalash

Obyektga mo‘ljallangan tahlil va loyihalash (object-oriented analysis and design) ning bosh g‘oyasi predmetga oid sohani va masalaning mantiqiy yechimini obyektlar (tushunchalar va mohiyatlar) nuqtayi nazaridan ko‘rib chiqishdan iborat. Obyektga mo‘ljallangan tahlil jarayonida asosiy diqqat-e’tibor obyektlar (yoki tushunchalar)ning predmetga oid soha atamalarida ta’riflash va tavsiflashga qaratiladi. Obyektga mo‘ljallangan tahlil jarayonida obyektga mo‘ljallangan dasturlash tili vositalari bilan joriy qilinadigan mantiqiy dasturiy obyektlar aniqlanadi. Bu dasturiy obyektlar atributlar va metodlarni o‘z ichiga oladi. Va, nihoyat, *konstruksiyalash (construction) yoki obyektga mo‘ljallangan dasturlash (object-oriented programming)* jarayonida ishlab chiqilgan komponentlar va sinflarning joriy qilinishi ta’minlanadi.

Obyektga mo‘ljallangan tahlil va loyihalashning ayrim asosiy tamoyillarini qisqacha ko‘rib chiqamiz. Keyinchalik biz barcha ko‘rib chiqilgan atamalarning yanada aniqroq ta’rifini va yanada to‘liqroq shifr yechimini beramiz.

–Avval *talablar tahlili (requirements analysis)* amalga oshiriladi hamda bu paytda modellashtirilayotgan tizimda sodir bo‘layotgan asosiy jarayonlar va ularning pretsedentlar ko‘rinishidagi ta’rifi ajratiladi. *Pretsedent (precedent)* – predmetga oid sohada sodir bo‘layotgan jarayonlarning matniy tavsifi.

– Ikkinchi qadam. *Predmetga oid sohaning obyektga mo‘ljallangan tahlili (object-oriented domain analysis)*. Bu qadamning vazifasi jarayon ishtirokchilarining faoliyat turlarini aniqlash hamda predmetga oid soha elementlarining turli kategoriyalarini aks ettiruvchi *konseptual model (conceptual model)* ni tuzishdan iborat.

– Uchinchi qadam. Bunda kim nima bilan shug‘ullanayotganini ko‘rib chiqamiz. Aynan shu faoliyat *obyektga mo‘ljallangan loyihalash (object-oriented design)* deb ataladi va bunda asosiy diqqat-e’tibor majburiyatlarni taqsimlashga qaratiladi. *Majburiyatlarni taqsimlash (responsibility assignment)* da ilovadagi turli dasturiy obyektlarning vazifa va majburiyatlari ajratib ko‘rsatiladi.

Obyektga mo‘ljallangan tahlil va loyihalashning muhim momenti shundan iboratki, bunda dasturiy tizim komponentlari o‘rtasida majburiyatlar taqsimotini malakali o‘tkazish ko‘zda tutiladi. Gap shundaki, obyektga mo‘ljallangan tahlil va loyihalashsiz ish ko‘rib bo‘lmaydi. Buning ustiga u robastlik, masshtablashish, kengayishlik

va takroran qo‘llash imkoniyatiga hal qiluvchi ta’sir ko‘rsatadi. Obyektlarning majburiyatlari va o‘zaro aloqalari *sinflar diagrammasi (design class diagram)* va *o‘zaro aloqalar diagrammasi (collaboration diagram)* qo‘llangan holda ifodalanadi.

1.5. Unifikatsiyalangan modellash tili — UMLga kirish

UML (Uniform Modeling Language) vizual modellashtirish tili bo‘lib, tizimlash arxitektorlariga tizim qay darajada standart va tushunish uchun oson shaklga ega ekani haqidagi o‘z tasavvurlarini namoyon etishga imkon beradi. UML loyihaviy yechimlar va ishlab chiquvchilarning o‘zaro aloqalarini qo‘shgan holda birgalikda foydalanishning samarali mexanizmini taqdim etadi.

Atamalar

— Diagramma — elementlar va ular o‘rtasidagi bog‘lanishlarning grafik tasviri.

— Tizim — qo‘yilgan vazifaning bajarilishini ta’minlaydigan dasturiy va apparat vositalar kombinatsiyasi.

— Tizimni ishlab chiqish deganda uni mijoz uchun, ya’ni biror bir muammoni hal qilishi lozim bo‘lgan inson uchun yaratish jarayoni tushuniladi.

UML ning foydasi

Analitik (tahlilchi) muayyan muammo tavsifi berilgan hujjatlarni ishlab chiqadi hamda ularni ishlab chiquvchilar — dasturchilarga beradi. Dasturchilar talab qilinayotgan masala yechimi uchun dasturiy ta’minot tayyorlaydi hamda uning apparat vositalarida yoyilishini kafolatlaydi.

Tasavvurdagi tizimni ifodalab berish tizim ustida ishlash jarayonining g‘oyat muhim bosqichidir. Avval tahlil «barmoqda sanab», ya’ni tavakkalchilik asosida o‘tkazilar edi. Hozirgi paytda ishlab chiqishning hal qiluvchi bosqichi bu puxta ishlangan rejadir. O‘z navbatida, reja ham mijoz talablari puxta tahlil qilib chiqilgach, tuzilishi kerak.

Loyihalash jarayonining muhim jihati bu jarayonni to‘g‘ri tashkil qilishdir. Bunda tizimni ishlab chiqishda ishtirok etayotgan analitiklar, mijozlar, dasturchilar va boshqa mutaxassislar bir-birini tushunib, umumiy fikrga kela olishlari talab qilinadi.

Zamonaviy tizimlarni ishlab chiqish jarayonining yana bir o‘ziga xos jihati shundaki, ishlarni bajarish uchun vaqt yetishmaydi. Agar

tarmoq tizimlarning chegaraviy topshirish muddatlari bir-biriga yaqin bo‘lsa, ishlab chiqish jarayonining uzluksizligini ta’minlash hayotiy muhim zaruratga aylanadi.

Hozirgi zamonning yana bir o‘ziga xos jihati — korporatsiyalarning birlashishi — ham ishlab chiqish jarayoniga o‘z talablarini qo‘yadi.

Tarix

UML ning mualliflari Grady Booch, James Rumbaugh va Ivar Jacobsonlardir. 80-yillar va 90-yillarning boshlarida ular bir-birlaridan mustaqil ravishda obyektga mo‘ljallangan tahlil va loyihalashning metodologiyasini o‘ylab topdilar. 1994–95-yillarda ular uchalasi Rational Software Corporation da birga bo‘ldilar.

UML ning dastlabki versiyalari dasturiy ta’minot tayyorlash sohasida qo‘llana boshladi, iste’molchilarning bildirgan fikrlari asosida esa ularga jiddiy o‘zgartirishlar kiritilib bordi. Bu UML (DEC, Hewlett-Parkard, Microsoft, Ration, ...) konsorsiumining yuzaga kelishiga zamin bo‘ldi. 1997-yilda konsorsium UML ning dastlabki versiyasini qabul qildi hamda uni ko‘rib chiqish uchun OMG ga taqdim etdi.

1997-yilning oxirida 1.0 versiyasi chiqdi. Shundan so‘ng OMG guruhi kuzatib borishga kirishdi hamda 1998-yilning oxirida uning yana ikkita yangi versiyasini chiqardi. UML tili dasturiy ta’minot ishlab chiqish sohasida de-fakto standarti bo‘lib qoldi. Hozirgi paytda til faol rivojlanishda davom etmoqda. 2002-yilda joriy hisoblanib kelayotgan 2.0 versiyasi chiqdi.

1.6. UML asoslari

Boshqa har qanday til kabi, UML ham lug‘at va qoidalardan iborat bo‘lib, ular UML tiliga kiritilayotgan so‘zlarni kombinatsiyalab, mazmunli konstruksiyalar olish imkonini beradi. Modellashtirish tilida lug‘at va qoidalar tizimni konseptual va jismoniy jihatdan taqdim etishga yo‘naltirilgan. UML ga o‘xshash modellashtirish tili dasturiy ta’minotning «chizmalari»ni tuzish uchun standart vosita bo‘lib xizmat qiladi.

UML lug‘ati asosiy konstruksiyalarning uchta turini o‘z ichiga oladi:

— *mohiyatlar* — modelning asosiy elementlari bo‘lgan abstraksiyalar;

— *munosabatlar* — mohiyatlar o‘rtasidagi aloqalar;

— ko‘plab mohiyatlar va munosabatlarning manfaatlarini ko‘zlovchi, guruhlashtiruvchi *diagrammalar.*

Endi ular haqida batafsilroq so‘z yuritamiz.

UML mohiyatlari

UML da mohiyatlarning to‘rtta turi mavjud:

— tuzilmaviy;

— xulqiy;

— guruhlashtiruvchi;

— annotatsiyali.

Mohiyatlar tilning obyektga mo‘ljallangan asosiy elementlaridir. Ularning yordamida to‘g‘ri modellarni yaratish mumkin. Tuzilmaviy mohiyatlar — bu UML tilidagi otlarning modeli. Odatda, bular modelning statik qismlari bo‘lib, tizimning konseptual yoki jismoniy elementlariga mos keladi. Yuqorida aytib o‘tilganidek, tuzilmaviy mohiyatlarning yettita turli ko‘rinishi mavjud bo‘lib, tabiiyki, ularning hammasi UML da o‘z aksini topdi. Tuzilmaviy mohiyatlarning ta’riflarini eslatib o‘tamiz hamda ularga mos keladigan UML grafik obrazining tavsifini keltiramiz.

Sinf (class) — atributlari, operatsiyalari, munosabatlari va ma’nolari umumiy bo‘lgan obyektlar majmuining tavsifi. Grafik jihatdan sinf to‘g‘ri to‘rtburchak ko‘rinishida ifodalanib, unda ushbu sinfning nomi, atributlari va operatsiyalari yozilgan bo‘ladi.

Interfeys (interface) — sinf yoki komponent (tarkibiy qism) taqdim etadigan ma’lum xizmatlar (servis, xizmatlar to‘plami) ni belgilaydigan operatsiyalar majmui. Diagrammalarda interfeys ostida ushbu interfeysning nomi yozilgan doira ko‘rinishida tasvirlanadi. Interfeys g‘oyat kam hollarda (deyarli hech bir holda desa ham bo‘ladi) o‘z holicha mavjud bo‘lmaydi, odatda u o‘zini joriy qiladigan sinf yoki komponentga qo‘shilib keladi.

Kooperatsiya (collaboration) o‘zaro aloqalarni belgilaydi. U turli rollar va boshqa elementlardan iborat bo‘lib, bu rol va elementlar birgalikda ishlab, qandaydir kooperativ effekt sodir qiladi. Grafik tarzda kooperatsiya punktir chiziqlar bilan chegaralangan ellips ko‘rinishida tasvirlanadi, ichida faqat uning nomi yozilgan bo‘ladi.

Pretsedent (use case) — bu tizim bajarayotgan xatti-harakatlar ketma-ketligining tavsifi bo‘lib, ushbu ketma-ketlik ma’lum bir aktyor (actor) uchun muhim bo‘lgan kuzatsa bo‘ladigan natija beradi. Grafik jihatdan pretsedent ham uzluksiz chiziq bilan chegaralangan ellips shaklida ifodalanib, uning ichida faqat pretsedent nomi yoziladi.

Faol sinf (active class) deb obyektlari bir yoki bir necha jarayonlarga, yoki iplar (threads) ga jalb qilingan, shuning evaziga boshqaruvchilik ta'sirini ko'rsata oladigan sinflarga aytiladi. Grafik jihatdan faol sinf oddiy sinf kabi tasvirlanadi, faqat yo'g'on chiziq bilan chegaralangan to'g'ri to'rtburchak shaklida bo'ladi, ichida nomi, atributlari va operatsiyalar yozilgan bo'ladi.

Komponent (component) yoki tarkibiy qism tizimning jismoniy o'rin almashadigan qismi bo'lib, u ma'lum bir interfeyslar yig'indisiga mos keladi hamda uning joriy qilinishini ta'minlaydi. Grafik jihatdan komponent ichida faqat nomi yozilgan qistirmali to'g'ri to'rtburchak ko'rinishida tasvirlanadi.

Uzel (node) — tizimning dasturiy mahsulot faoliyat ko'rsatayotgan vaqtda mavjud bo'lgan elementi. Bu element ma'lum bir xotira hajmiga, shuningdek, ishlov berish imkoniyatiga ega bo'lgan hisoblash resursidir. Grafik tasviri kub ko'rinishida bo'lib, ichida uzelning nomi yoziladi, xolos.

Yuqorida sanab o'tilgan yettita bazaviy element (sinflar, interfeyslar, kooperatsiyalar, pretsedentlar, faol sinflar, komponentlar va uzellar) UML modelida qo'llanishi mumkin bo'lgan asosiy tuzilmaviy mohiyatlardir. Mohiyatlarning boshqa turlari ham mavjud: aktyorlar, signallar, utilitlar (sinflar turlari), jarayonlar va iplar (faol sinf turlari), ilovalar, hujjatlar, fayllar, kutubxonalar, sahifalar va jadvallar (komponentlar turlari).

Endi mohiyatlarning biz avval aytib o'tmagan boshqa turlarini aniqlaymiz.

Xulqiy mohiyatlar (behavioral trings) UML modelining dinamik tashkil qiluvchisidir. Bu tildagi fe'llardir. Ular modelning vaqt va fazodagi xulq-atvorini tavsiflaydi. Xulqiy mohiyatlarning faqat ikkita turi mavjud:

O'zaro aloqa (interaction) xulq-atvor bo'lib, uning mohiyati ma'lum maqsadga erishish uchun aniq bir kontekst chegarasida obyektlar o'rtasidagi ma'lumotlar (messages) almashinuvidan iborat. O'zaro aloqa yordamida alohida operatsiyani ham, obyektlar to'plamining xulq-atvorini ham tavsiflash mumkin. O'zaro aloqa xabarlar, xatti-harakatlar ketma-ketligi va aloqalar (obyektlar o'rtasida) kabi boshqa elementlarning bo'lishini ham talab qiladi. Grafik jihatdan xabar strelka ko'rinishida tasvirlanib, uning tepasida deyarli hammavaqt tegishli operatsiyaning nomi yozilgan bo'ladi.

Avtomat (state machine) holatlar ketma-ketligini belgilovchi xulq-atvor bo'lib, obyektlar yoki o'zaro aloqalar o'z yashash davri davomida turli voqea-hodisalarga javoban ushbu holatlardan o'tadi

yoki ularga o‘z munosabatini bildiradi. Avtomatlar yordamida alohida sinf yoki sinflar kooperatsiyasining xulq-atvori tavsiflanadi. Avtomat bilan qator boshqa elementlar ham bog‘liq. Bular: holatlar, bir holatdan boshqasiga o‘tishlar, voqea-hodisalar — xatti-harakatlarning o‘tishini va turlarini nomlovchi mohiyatlar — o‘tishlarga javoblar. Grafik jihatdan holatlar burchaklari sal yumaloqlangan to‘g‘ri to‘rtburchak shaklida tasvirlanadi, ichiga holat nomi yoziladi.

O‘zaro aloqalar va avtomatlar UML modeliga kiruvchi asosiy xulqiy mohiyatlardir. Semantik (ma’no) jihatdan ular turli tuzilmaviy elementlar, birinchi navbatda, sinflar, kooperatsiyalar va obyektlar bilan bog‘langan bo‘ladi.

Guruhlanuvchi mohiyatlar UML modelining tashkil qiluvchi qismlaridir. Bular modelni yoyib qo‘yish mumkin bo‘lgan bloklardir. Bunday birlamchi mohiyat bitta nusxada mavjud bo‘ladi. Bu — paket.

Paketlar (packages) — elementlarni guruhlarda tashkil qiluvchi universal mexanizm. Paketga tuzilmaviy, xulqiy va boshqa turdagi guruhlovchi mohiyatlarni solib qo‘yish mumkin. Dastur ishlayotgan paytda haqiqatan mavjud bo‘lgan komponentlardan farqli o‘laroq paketlar sof konseptual xarakterga ega, ya’ni faqat ishlab chiqish jarayonida mavjud bo‘ladi. Paketni tasvirlash uchun papkaning qistirmali piktogrammasi qo‘llanadi va unda faqat papka nomi, ba’zida esa uning ichidagilar ko‘rsatilgan bo‘ladi.

Annotatsiyali mohiyatlar — UML modelining izohlovchi qismi. U modelning har qanday elementiga qo‘shimcha tavsif, ta’rif yoki tanbehlar uchun sharhdan iborat. Annotatsiyali elementlarning faqat bitta bazaviy turi mavjud — izoh.

Izoh (note) — sharh yoki cheklashlarni tasvirlash uchun element yoki elementlar guruhiga qo‘shilgan oddiy simvol. Grafik jihatdan ilova chekkasi qayrilgan to‘g‘ri to‘rtburchak ko‘rinishida tasvirlanadi hamda matniy yoki grafik sharhga ega bo‘ladi.

Bu element siz UML modelida qo‘llashingiz mumkin bo‘lgan asosiy annotatsiyali mohiyat hisoblanadi. Ko‘p o‘rinda ilovalardan diagrammalarni noformal yoki formal matn ko‘rinishida ifodalasa bo‘ladigan sharhlar yoki cheklashlar bilan ta’minlash uchun foydalaniladi. Bu elementning variantlari, masalan, talablar mavjud bo‘lib, ularda tizimning modelga nisbatan tashqi istakdagi xulq-atvori tavsiflanadi.

UML ning o‘zaro munosabatlari

UML tilida munosabatlarning to‘rtta turi belgilangan:

— tobelik;

— assotsiatsiya;
— umumlashtirish;
— joriy etish.

Bu munosabatlar UML da asosiy bog‘lovchi konstruksiyalar hisoblanadi hamda to‘g‘ri modellarni qurishda qo‘llanadi. Endi aytilganlarni tartib bilan bayon qilsak.

Tobelik (dependency) ikkita mohiyat o‘rtasidagi semantik (ma’no jihatdan) munosabat bo‘lib, bunda ulardan birining, tobe bo‘lmaganining o‘zgarishi boshqasining, tobe bo‘lganining o‘zgarishiga ta’sir ko‘rsatishi mumkin. Grafik jihatdan tobelik punktir chiziq bilan ifodalanadi, bu chiziq, odatda, belgisi bo‘lgan strelkaga ega bo‘ladi.

Assotsiatsiya (association) tuzilmaviy munosabat bo‘lib, aloqalar majmuini tavsiflaydi. Bunda aloqa deganda obyektlar o‘rtasidagi biror bir ma’noli aloqa tushuniladi. *Agregatlash (aggregation)* assotsiatsiyaning ko‘rinishlaridan biridir. Butun va uning qismlari o‘rtasidagi tuzilmaviy munosabat shunday ataladi. Grafik jihatdan assotsiatsiya chiziq ko‘rinishida ifodalanadi (ba’zida bu chiziq strelka bilan tugallanadi yoki biron belgiga ega bo‘ladi). Chiziq yonida qo‘shimcha belgilar ham bo‘lishi mumkin, masalan, karralilik va rollarning nomi.

Umumlashtirish (generalization) — bu «ixtisoslashuv/umumlashtirish» munosabati. Bunda ixtisoslashgan element obyekti (oddiyroq qilib aytganda, avlod) umumlashtirilgan element (ajdod) obyekti o‘rniga qo‘yilishi mumkin. Obyektga mo‘ljallangan dasturlashda qabul qilinganidek, *avlod (childe)* o‘z *ajdodi (parent)* ning tuzilishi va xulq-atvorini meros qilib oladi. Umumlashtirish munosabati grafik jihatdan bo‘yalmagan strelkali chiziq ko‘rinishida ifodalanadi. Strelka ajdodni ko‘rsatib turadi.

Va, nihoyat, *joriy etish (realizatsion)* klassifikatorlar o‘rtasidagi semantik munosabat. Bunda bitta klassifikator majburiyatni belgilaydi, ikkinchisi uning bajarilishini kafolatlaydi. Joriy etish munosabatlari ikkita vaziyatda uchraydi: birinchidan, interfeyslar hamda ularni joriy qiluvchi sinflar yoki komponentlar o‘rtasida, ikkinchidan, pretsedentlar hamda ularni joriy qiluvchi kooperatsiyalar o‘rtasida. Joriy qilish munosabatlari bo‘yalmagan strelkali punktir chiziq ko‘rinishida ifodalanadi.

Biz UML ning to‘rtta elementining tavsifini berdik. Ular UML modellarida asosiy munosabat turlari bo‘lib xizmat qiladi. Ularning variantlari ham mavjud bo‘lib, ulardan biri, masalan, *aniqlashtirish (refinement), trassalash (trace)*, tobeliklar uchun *ulanish* va *kengaytirish*.

1.7. UML diagrammalari

UML tili diagrammalarda qo‘llanadigan grafik elementlar to‘plamini o‘z ichiga oladi. Til bo‘lgani uchun UML ushbu elementlarni birlashtiruvchi qoidalarga ega.

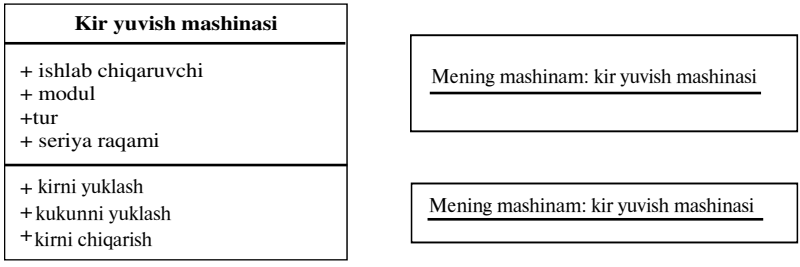
Diagrammalar tizimning turli tasvirlarini aks ettirish uchun qo‘llanadi. Turli tasvirlarning ushbu to‘plami model deb ataladi. UML tizimi modelini binoning badiiy bezatilgan modeli bilan qiyoslash mumkin. Shuni ta’kidlab o‘tish muhimki, UML modeli tizim nima qilishi lozimligini tavsiflaydi. Shu vaqtning o‘zida bu qanday joriy qilinishi ma’lum qilinmaydi.

Umuman olganda, modelni yaratishda yaxshi ma’lum bo‘lgan nimadir qo‘llanadiki, bundan maqsad yaxshi ma’lum bo‘lmagan nimanidir tushunib yetish.

Sinflar diagrammasi

Bizni o‘rab turgan barcha buyumlar kategoriyalari bo‘yicha ajralib turishini sezish qiyin emas (avtomobillar, mebel, kir yuvish mashinasi). Biz bu kategoriyalarga sinflar sifatida murojaat qilamiz. Sinf — bu o‘xshash atributlar va umumiy xususiyatlarga ega bo‘lgan buyumlar kategoriyasi yoki guruhi.

Sinflar diagrammasi ishlab chiqish jarayonining boshlang‘ich nuqtasidir.

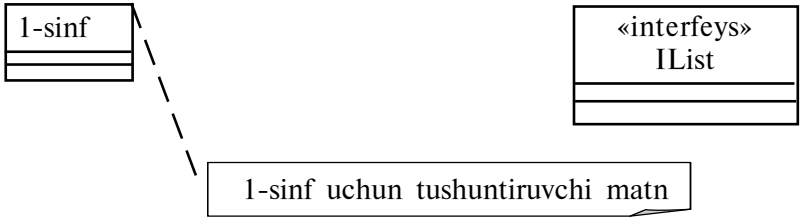


Obyektlar diagrammasi

Obyekt sinf nusxasi bo‘lib, u atributlar va operatsiyalarning avvaldan berilgan qiymatlariga ega. Agar kir yuvish mashinasi misolida ko‘rib chiqsak, bu holda uning atributlari quyidagi ko‘rinishga ega bo‘ladi: ishlab chiqaruvchi kompaniya — «Сантехника», model nomi — «Мойдодыр», servis raqami — «13-666-13» va hajmi — 16 funt.

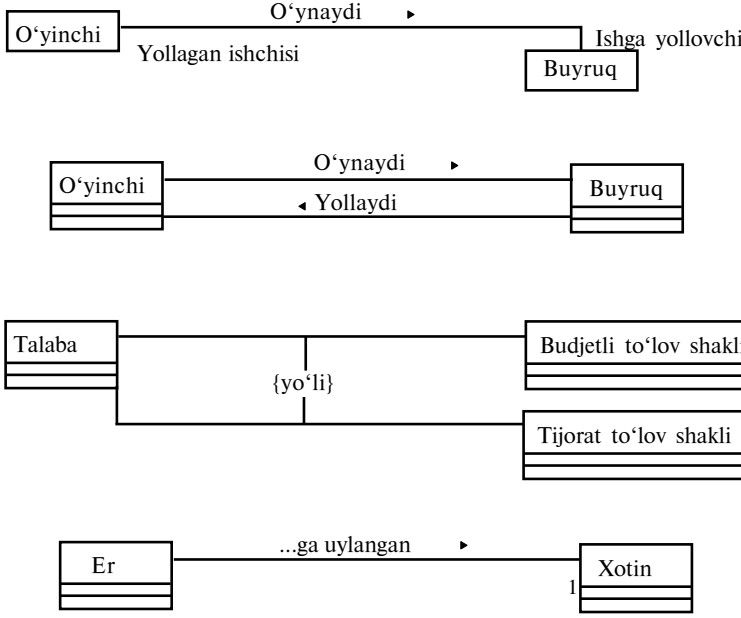
Tilning kengayishi

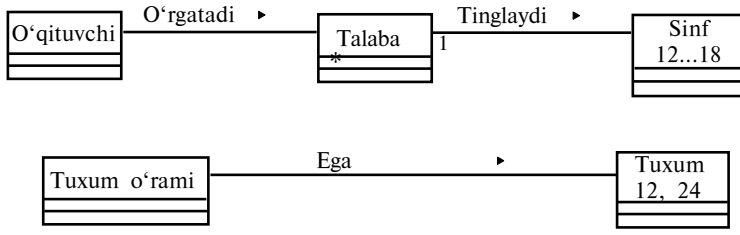
Izohlar nimaga xizmat qiladi? Diagrammaning ushbu qismi nima uchun aynan shu yerda joylashgan va u bilan qanday ishlash kerak, degan savollarga javob beradi. Stereotiplar UML ning mavjud elementlarini qo‘llash va qayta o‘zgartirish imkonini beradi. Interfeys konsepsiyasi, ya’ni atributlarga ega bo‘lmagan sinf bunga yaxshi misol bo‘la oladi.



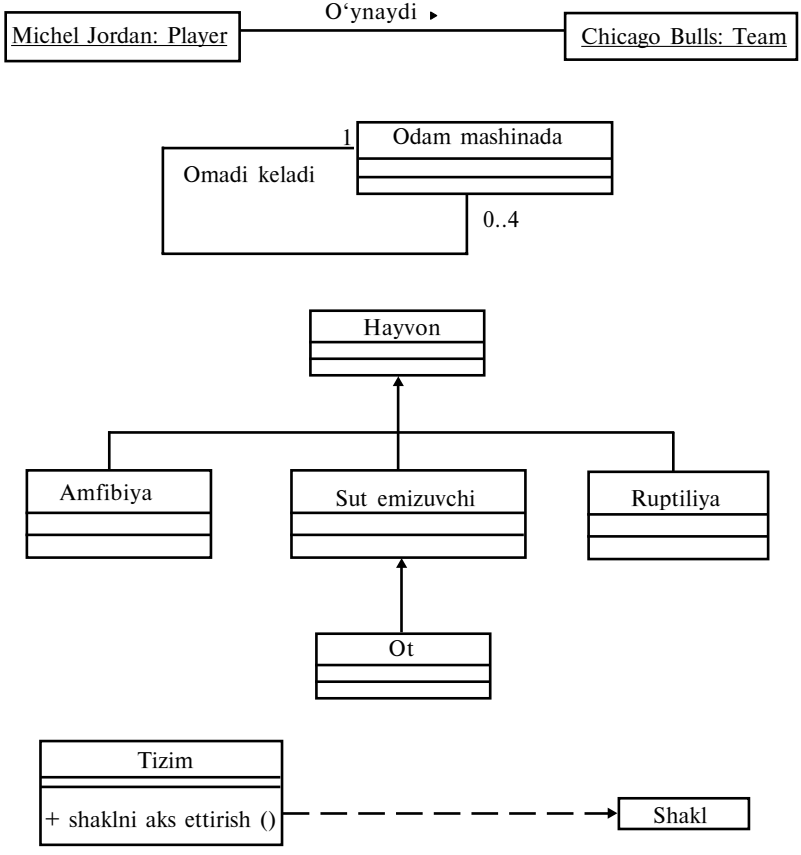
Assotsiatsiyalar

Agar sinflar konseptual jihatdan bir-biri bilan aloqa qilsa, bu aloqa assotsiatsiya deb ataladi. Assotsiatsiyalar cheklagichlarga (masalan{navbat bo‘yicha}, {yoki}), kvalifikatorga («bittasi ko‘pga» nisbatan qo‘shimcha axborot), sinflarga, bo‘lishlilikka ega, refleksiv bo‘lishi mumkin.



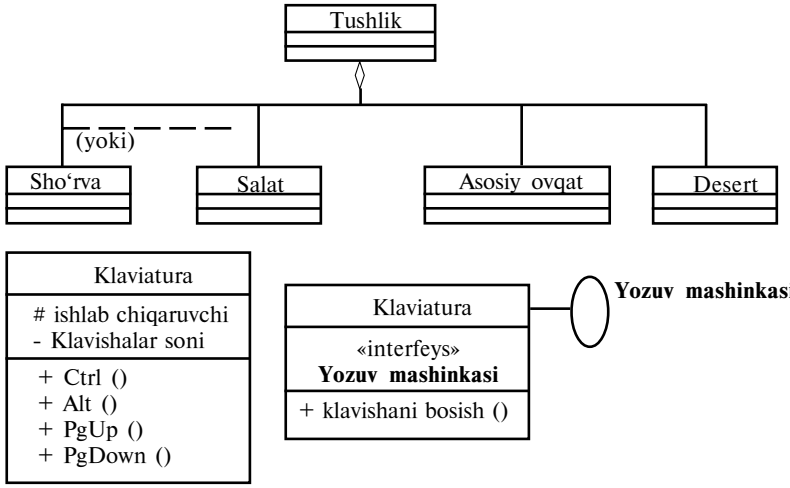


Vorislik, tobelik bo‘lishi mumkin.



Agregatlash, kompozit obyektlar, interfeyslar va joriy qilish

Ba’zida sinf komponent-sinflardan tashkil topadi. Bu o‘zaro aloqaning agregatsiya deb ataluvchi alohida turi. Tanlovni ko‘rsatish maqsadida {yoki} dan foydalanish mumkin. Agregatsiya «qism-butun» munosabatlarini taqozo etadi.



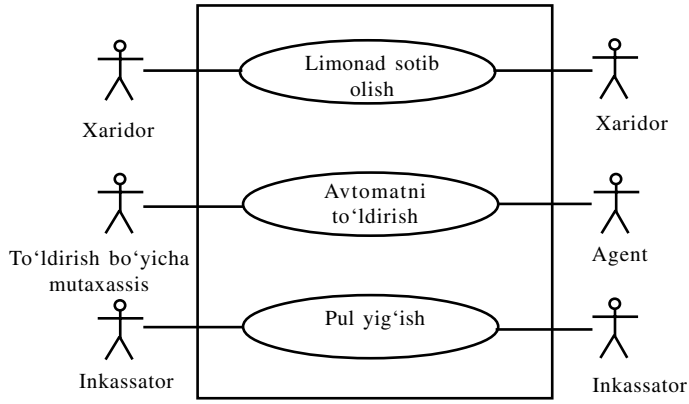
Kompozit agregatsiyaning asl turi bo‘lib, har bir element faqat bitta butunga tegishli ekani bilan tavsiflanadi. Takroran qo‘llanadigan operatsiyalar to‘plamini ajratib olish va bir guruhga birlashtirish mumkin. Interfeysni belgilashning ikkita turi mavjud: doiracha va stereotip. «Ko‘rimlilik» atamasi atributlar va operatsiyalarga nisbatan qo‘llanadi hamda ulardan foydalanishi mumkin bo‘lgan boshqa sinflar turlarini keltirib chiqaradi.

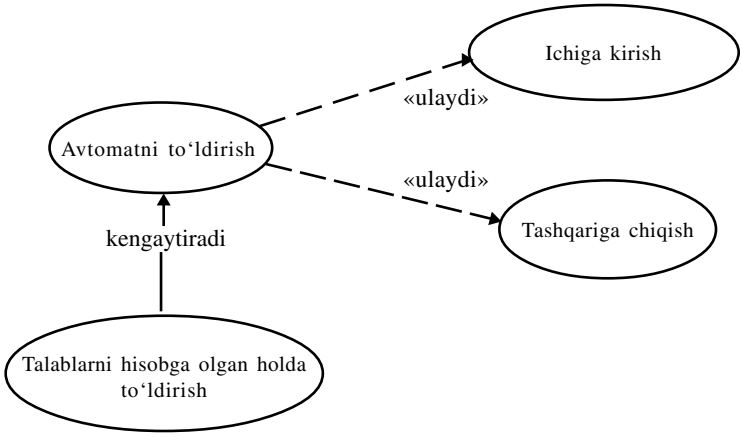
- Ochiq soha «+» — hamma foydalanishi mumkin.
- Himoyalangan soha «#» — faqat merosxo‘rlar tomonidan qo‘llanadi.
- Yopiq soha «—» — faqat sinf kirish huquqiga ega.
- Joriy qilish interfeys operatsiyalarining ochiqligini ko‘zda tutadi.
- Statik atributlar va operatsiyalar (sinfga xos, ushbu sinfning barcha obyektlari uchun yagona) ning tagiga chiziladi.

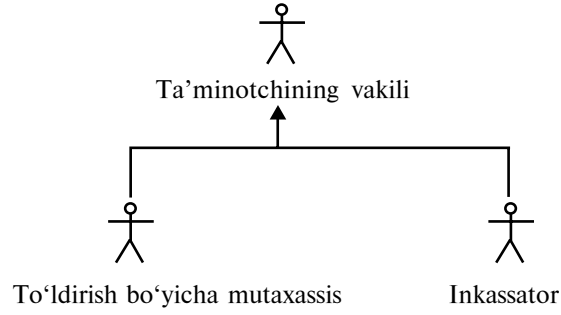
Pretsedentlar diagrammasi (Use case)

Pretsedent — tizimdan foydalanish ssenariylarining to‘plami. Har bir xatti-harakatlar ketma-ketligi boshqa tizim, foydalanuvchi va h.k. tomonidan vaqtning qandaydir daqiqasida nomlanadi. Mohiyatlar, nomlanuvchi ssenariylar bajaruvchi deb ataladi. Pretsedentlardan takroran foydalanish mumkin. Usullardan biri ulash, ikkinchisi kengaytirish.

Pretsedentlarni, sinflar kabi, umumlashtirish (vorislik qilish) mumkin. Vorislik qilishda sho‘ba pretsedent ajdodining pretsedentiga o‘z qadamlarini qo‘shadi. Bajaruvchilar ham nasldan naslga o‘tishi mumkin. Boshida yuqori darajali pretsedentlar diagrammasini yaratish g‘oyat muhimdir.





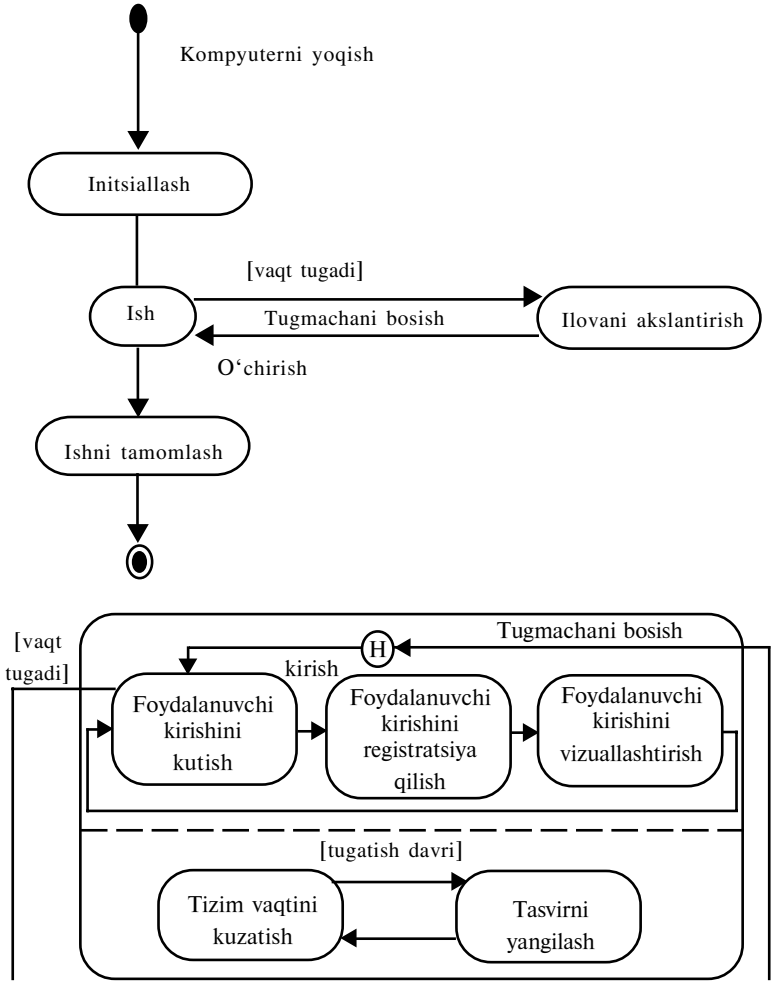


Holatlar diagrammalari

Sodir bo‘layotgan voqealarga javoban hamda vaqt o‘tishi bilan obyektlar o‘z holatlarini o‘zgartiradi. Holatlar diagrammasi obyekt

holatlari hamda ular o‘rtasidagi o‘tishlarni, shuningdek, obyektning dastlabki va oxirgi holatini ko‘rsatadi.

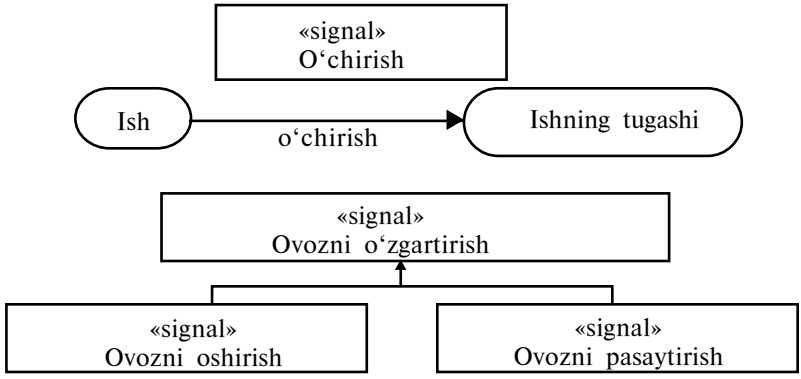
Taymer va schotchiklar kabi holatlar o‘zgaruvchilari ba’zida g‘oyat foydali bo‘lishi mumkin. Faoliyat turlari voqealar va xatti-harakatlardan iborat bo‘ladi. Kirish (tizim holatga kirganda nima sodir bo‘lishi), chiqish (tizim holatdan chiqqanda nima sodir bo‘lishi), bajarish (tizim holatda turganda nima sodir bo‘ladi) faoliyatning standart turlariga kiradi.



O‘tish liniyalariga qo‘shimcha detallarni qo‘shish mumkin. Masalan, o‘tishni keltirib chiqargan voqea ko‘rsatiladi. Holatlar murakkab

bo‘lishi mumkin. Ular o‘z ichida tarmoq tizimlar (hatto tarmoq tizimlar guruhi)ga ega bo‘lishi mumkin. Bunday holatlar kompozit holatlar deb ataladi. Bu doiracha ichiga yozilgan «H» (History) harfi bilan belgilanadi. Barcha tobe holatlar xotirada saqlangan chuqur tarix uchun «H*» belgisi qo‘yiladi.

Holatlar diagrammasida qabul qiluvchi obyektning o‘tishini ta’minlovchi voqea signal deb ataladi. Boshqa har qanday sinflar kabi, signallarni ham meros olish mumkin. O‘tish shartli (voqea bo‘yicha) yoki shartsiz (barcha xatti-harakatlarning bajarilishi bo‘yicha) bo‘lishi mumkin.



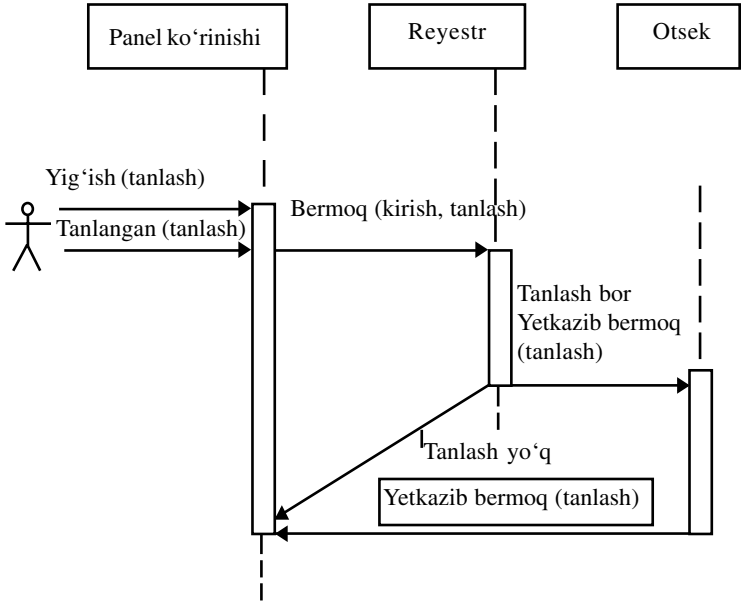
Holatlar diagrammasida tizimdagi bitta obyekt holatlari o‘rtasidagi barcha o‘tishlar o‘z aksini topadi. Diagrammalar analitiklar (tahlilchilar) va developerlarga xohishdagi xulq-atvor haqida to‘liq axborot beradi.

Ketma-ketliklar diagrammalari

Ketma-ketliklar diagrammasi oddiy obyektlardan, xabarlardan (strelkalar ko‘rinishida), shuningdek, vertikal vaqt o‘qidan iborat. Xabar oddiy (boshqaruvni uzatish), sinxron (javob kutadi), nosinxron (javob kutmaydi, amal qilishda davom etadi) bo‘lishi mumkin. Misol: 1. Xaridor tangani avtomatning old panelidagi teshikka joylaydi. 2. Xaridor limonad navini tanlaydi. 3. Tanga reyestrga kelib tushadi. 4. Ko‘rib chiqilayotgan asosiy ssenariy uchun limonadning kerakli navi mavjud hamda reyestr limonadni avtomatning old paneliga yetkazib berish uchun buyruq beradi, deb hisoblaymiz.

Ketma-ketliklar diagrammasida quyidagi holatlar sodir bo‘lishi mumkin: [shart bo‘yicha o‘tish], *[hozircha sikli], yaratish (yaratmoq(), ′<<′ yaratmoq ′>>′), obyektni bartaraf etish (X).

UML ning ketma-ketliklar diagrammasi obyektlarning o‘zaro aloqasiga vaqt o‘zgarishini ham qo‘shadi. Grafikda u uzunchoq tor to‘g‘ri to‘rtburchak ko‘rinishida ifodalanadi va aktivatsiya nuqtasi (operatsiyalardan birining bajarilishi) ni ko‘rsatadi.



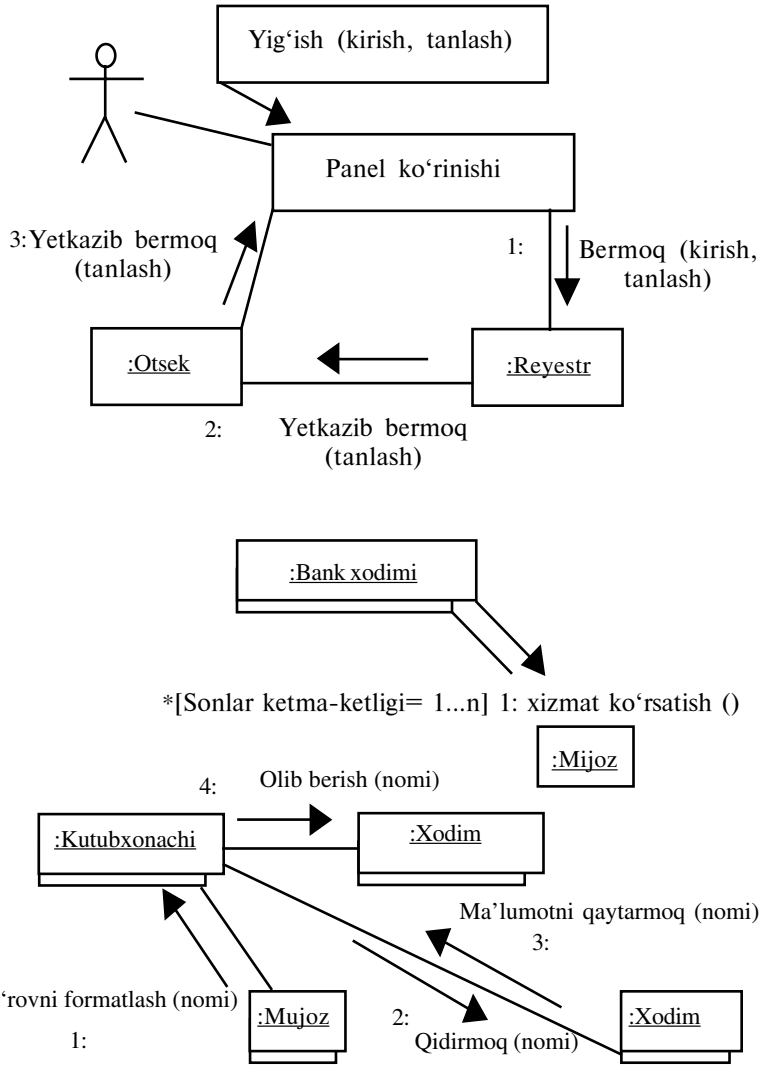
Kooperatsiya diagrammalari

Avval ketma-ketliklar diagrammasida tasvirlangan axborotni kooperatsiya diagrammasida ham ifodalash mumkin. Diagrammalarning bu ikkita turi semantik (ma’no) jihatdan ekvivalentdir. Ketma-ketliklar diagrammasi vaqtga muvofiq tartiblashtirilgan bo‘lsa, kooperatsiya diagrammasi obyektlarning fazodagi joylashuviga muvofiq tartiblashtirilgan.

Kooperatsiya diagrammasida obyektlar o‘rtasidagi assotsiatsiya bio-obyekt ikkinchisiga uzatadigan xabar ko‘rinishida ifodalangan. Xabarning ifodalanishi: strelka, tartib, raqam, nom. Shartlar, avval bo‘lganidek, kvadrat qavslarda aks ettiriladi. Sikl (davr)ga tavsif berish uchun shart oldidan yulduzcha qo‘yish kerak.

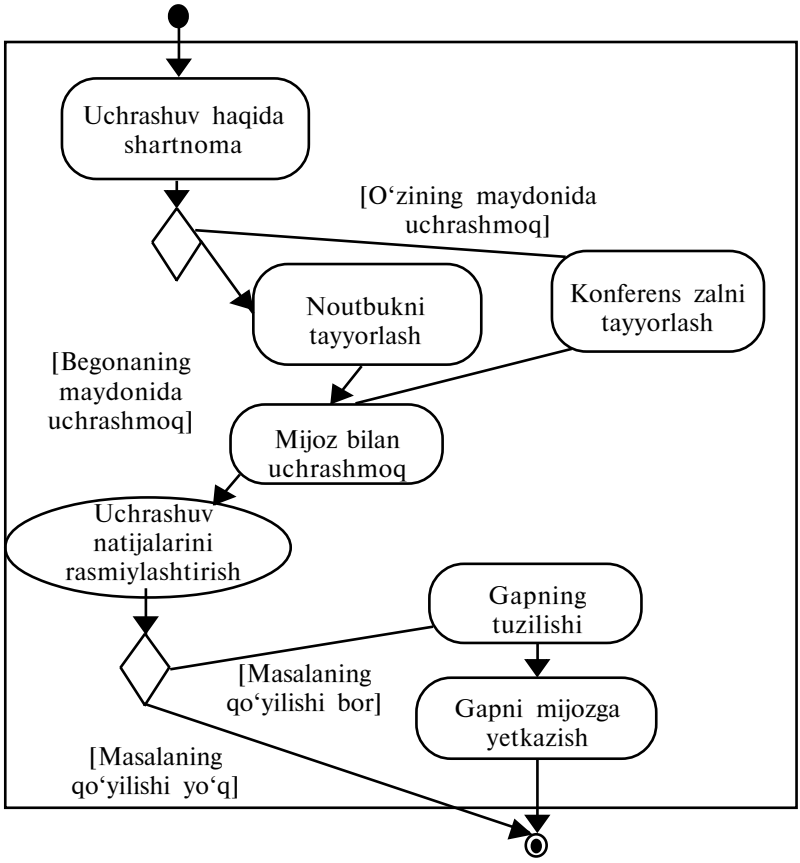
Ayrim operatsiyalar boshqalariga nisbatan sho‘ba operatsiyalari hisoblanadi. Xabarlarni raqamlash tizimida ularning raqami ajdod nomini bildiruvchi o‘nlik sondan keyin qo‘yilgan nuqtadan so‘ng yoziladi. Kooperatsiya diagrammasi qabul qiluvchilarning ko‘plab obyektlarini modellashtirish imkonini beradi. Xabarlar

oqimini boshqaruvchi faol obyektlarni, shuningdek, sinxronlashtirilgan xabarlarni ham tasvirlashi mumkin.



Faoliyat turlarining diagrammalari

Bu diagrammalar blok-sxemalarga g‘oyat o‘xshash. Bunda qarorlarni qabul qilish nuqtalari, baryerlar va parallel oqimlar, signallarni uzatish mavjud. Kerakli faoliyat turi qaysi obyektga tegishli ekanini spetsifikatsiyalash imkoniyati ham bor.



Komponentlar diagrammasi

Misollar: jadval, ma’lumotlar massivi, bajarilayotgan fayl, dinamik ravishda ulanadigan kutubxona, hujjat va h.k.

Komponentlar turlari:

1. Yoyib yuborish komponentlari, ular ishchi tizimlar bazisi (DLL, bajarilgan fayllar) ni shakllantiradi.

2. Faoliyat natijalari komponentlari, ulardan yoyib yuborish komponentlari (ma’lumotli fayllar) yaratiladi.

3. Bajarish komponentlari, ular tizimning ishlashi natijasida yaratiladi.

Yoyib yuborish diagrammalari

UML ning yoyib yuborish diagrammasi tizimni tayyor ko‘rinishda tavsiflaydi. Tizim uzellardan tashkil topib, uzellarning har bittasi kub ko‘rinishida ifodalanadi. Ikki kub o‘rtasidagi chiziq uzellarning ulanishini bildiradi.

Ikki turdagi uzellar mavjud: protsessor (u komponent buyruqlarini bajara oladi) va qurilma (komponent buyruqlarini bajara olmaydigan uzel).

2. C++GA KIRISH

2.1. C++da o‘zgaruvchilar turlari va tavsiflari

Har bir nom va har bir ifoda o‘z turiga ega bo‘lib, bu tur o‘z ustida amalga oshirilishi mumkin bo‘lgan operatsiyalarni belgilab beradi. Masalan, int i tavsifini olaylik. U i ning int turga ega ekanini, ya’ni i butun o‘zgaruvchi ekanini belgilaydi. Tavsif — dasturga nom kiritadigan operator. Tavsif ushbu nomga tur bag‘ishlaydi. Tur nomning yoki ifodaning to‘g‘ri qo‘llanishini belgilaydi. Butunlar uchun +, —, * va / kabi operatsiyalar belgilangan.

Asosiy turlar

Apparat ta’minoti vositalariga yaqindan bevosita javob beradigan asosiy turlar quyidagilardir: char short int long float double.

Dastlabki to‘rtta tur butun sonlarni, keyingi uchtasi esa suzuvchi nuqtali sonlarni, ya’ni kasrlarni ifodalashda qo‘llanadi.

Char turdagi o‘zgaruvchi ushbu mashinada simvolni saqlash uchun mo‘ljallangan o‘lchamga ega, int turdagi o‘zgaruvchining o‘lchamlari esa ushbu mashinadagi butun arifmetikaga mos keladi (bu, odatda, so‘z bo‘ladi). Tur taqdim etishi mumkin bo‘lgan butun sonlar diapazoni ushbu turning o‘lchamlariga bog‘liq bo‘ladi (uni sizeof operatori yordamida hisoblab chiqarish mumkin).

C++da o‘lchamlar char turdagi ma’lumotlar o‘lchamlari birligida o‘lchanadi, shuning uchun char ta’rifga binoan 1 o‘lchamiga ega. Asosiy turlar o‘rtasidagi nisbatni quyidagicha yozish mumkin:

1=sizeof(char)<=sizeof(short)<=sizeof(int)<=sizeof(long)<=sizeof(float)<=sizeof(double)

Umuman olganda, turlar haqida yana biron nimani taxmin qilish aqldan emas. Jumladan, butun son ko‘rsatkichni saqlash uchun yetarli ekani hamma mashinalar uchun ham birdek to‘g‘ri kelavermaydi. Asosiy turga nisbatan const sifatini qo‘llash mumkin. Bunda dastlabki turga xos bo‘lgan xususiyatlarga ega turni olish imkonini beradi. Bunda faqat bitta istisno bor. U shundaki, const turidagi o‘zgaruvchilarning qiymati initsiallash (nomlashtirish) dan so‘ng o‘zgara olmaydi.

```
const float pi=3.14;
const char plus='+'.
```

Bitta tishli qo‘shtirnoqqa olingan belgi (simvol) belgi konstantasi hisoblanadi.

E’tibor bering, shunday yo‘l bilan aniqlangan konstanta xotirani egallamaydi. Shunchaki, konstanta qiymati kerak bo‘lib qolgan o‘rinda u bevosita qo‘llanishi mumkin bo‘ladi. O‘zgaruvchilar uchun nomlantirish shart bo‘lmasa-da, biroq qat’iy tavsiya etiladi. Lokal o‘zgaruvchini nomlantirmay turib kiritishga deyarli asos yo‘q.

Ushbu turlarning har qanday kombinatsiyasiga quyidagi arifmetik operatsiyalarni qo‘llash mumkin:

+ (unar va binar plus)

— (unar va binar minus)

* (ko‘paytirish)

/ (bo‘lish)

Shuningdek, ularga quyidagi qiyoslash operatsiyalarini ham qo‘llash mumkin:

== (teng)

!=(teng emas)

<(kamroq)

>(ko‘proq)

<=(kamroq yoki teng)

>=(ko‘proq yoki teng)

Agar operatsiya o‘tkazilayotgan qiymatlar qo‘yilgan shartga javob bersa, qiyoslash operatsiyalari natijada 1 ga, aks holda 0 ga teng bo‘ladi.

E’tibor bering, butun taqsimlash butun natija beradi: 7/2 3 ga teng. Butun ustida qoldiqni olish foizi operatsiyasi o‘tkazilishi mumkin: 7%2 1 ga teng.

C++qiymat berishda va arifmetik operatsiyalarni o‘tkazishda asosiy turlar o‘rtasida, ularni cheklanmagan holda bir-biri bilan biriktirish uchun barcha aqliy qayta o‘zgartirishlarni bajaradi:

```
double d=1;
int i=1;
d=d+i;
i=d+i;
```

Satr turlari

C++da matn satrlarini saqlash uchun maxsus AnsiString ma’lumotlar turi qo‘llanadi. U belgilarning qandaydir to‘plami (ya’ni massivi)dan iborat bo‘ladi.

«Satr» turining o‘zgaruvchanlari xuddi har qanday boshqa o‘zgaruvchanlar kabi e’lon qilinadi va nomlashtiriladi.

Keyingi belgilar to‘plami yangi o‘zgaruvchining nomi emas, balki satr ekanini kompilyatorga uqtirish uchun satrni ikki tirnoqli qo‘shtirnoqqa olish kerak.

Misol:

AnsiString st=″matnning satri″.

Satr turidagi o‘zgaruvchi boshqa satr o‘zgaruvchisi bilan qo‘shish operatsiyasini bajarishga yo‘l qo‘yadi. Bu operatsiya ikkita satrning o‘zgaruvchilar kelgan tartibda qo‘shilishi sifatida qabul qilinadi.

Misol:

```
AnsiString s1=″satri″;
AnsiString s2=″matnning″;
AnsiString s=s1+s2;
```

Natijada s o‘zgaruvchisi tarkibida ′matnning satri′ qiymati mavjud bo‘ladi, u s1 va s2 o‘zgaruvchilarning qiymatlaridan tarkib topadi.

Qo‘shimcha turlar

Borland C++ da butun sonli o‘zgaruvchilar turlarining qo‘shimcha bo‘linishi sodir bo‘ladi. Bu yerda o‘zgaruvchilar turlarining barcha nomlari __intX turidagi operator hisoblanadi: bu yerda X — o‘zgaruvchi qiymat maydonining bitlarda ifodalangan o‘lchamlari. Bunda X quyidagi qiymatlarni olishi mumkin: 8, 16, 32 va 64. Bunday turdagi o‘zgaruvchilarni qo‘llash standart tur belgilab bergan o‘zgaruvchilarni qo‘llashdan farq qilmaydi.

Quyidagi jadvalda shunday turlar bilan ishlash ko‘rsatib berilgan:

Tur nomi	O‘zgaruvchini tavsiflashga misol	O‘lchamlar
__int8	__int8 c=128;	8 bit
__int16	__int16 s=32767;	16 bit
__int32	__int32 i=123456789;	32 bit
__int64	__int64 big= 12345654321;	64 bit
unsigned__int64	unsigned__int64 k=123456787654321;	64 bit

Standart turlarni qayta o‘zgartirish

Ma’lumotlar turlari C++tomonidan qattiq nazorat qilingani uchun turlarni qayta o‘zgartirishning imkon darajada qiymatlarni saqlaydigan operatsiyalarni bajarish ko‘zda tutiladi.

Boshqa turdagi o‘zgaruvchidan ma’lum turdagi o‘zgaruvchiga ega bo‘lish uchun o‘zgaruvchining quyidagi konstruksiyasi (yangi turi) qo‘llanadi.

Misol:

```
short S=100;
int I=(int)S;
```

Ushbu misol ortiqcha buyruqlarga ega, chunki C++da ko‘pchilik turlar o‘zgaruvchanlarining to‘g‘ridan-to‘g‘ri berilishi ko‘zda tutilgan, biroq ayrim hollarda (masalan, o‘zgaruvchi qiymatini biror bir funksiyaga berishda) bu buyruqlarning ahamiyati katta.

Sonli qiymatlarni starlarga aylantirish

C++turlari to‘g‘ridan-to‘g‘ri qayta o‘zgartirilgani tufayli o‘zgaruvchini shakl komponentlarining ko‘pchiligida qo‘llanadigan belgilar satri ko‘rinishidagi o‘nlikda ifodalab qayta o‘zgartirishga yo‘l qo‘ymaydi. To‘g‘ridan-to‘g‘ri qayta o‘zgartirish faqat asosiy va qo‘shimcha turlar uchun yo‘l qo‘yiladi, biroq massiv (ya’ni satr) kabi hosila turlar uchun yo‘l qo‘yilmaydi.

Bu kabi qayta o‘zgartirish uchun qayta o‘zgartirishning standart funksiyalari qo‘llanadi, bular intToStr, StrToInt, FloatToStr va boshqalar. Ma’lumotlar turlarining ko‘pchiligi uchun satrga va aksincha qayta o‘zgartirishning shunday funksiyasi mavjud.

Misol:

```
char S[10]; // belgilar massivi
int I=100; //butun sonli o‘zgaruvchi
S=intToStr(1); // qayta o‘zgartirish
```

2.2. Ko‘rsatkichlar va iqtiboslar bilan ishlash

Ko‘rsatkichlar

Ko‘rsatkich — xotira uyasining unikal raqamini saqlaydigan o‘zgaruvchi. Ko‘rsatkich operativ xotiradagi biror bir o‘zgaruvchi mavjud bo‘lishi mumkin bo‘lgan biror bir joyni belgilaydi. Ko‘rsatkichlarning qiymatlarini o‘zgartirish, turli variantlarda qo‘llash mumkinki, bu dasturga ko‘proq moslashuvchanlikni baxsh etadi.

Ko‘rsatkich odatda tiplangan bo‘ladi hamda quyidagicha e’lon qilinadi:

<turning nomi>*<ko‘rsatkichning nomi>=<dastlabki qiymat>

Ma’lum turdagi biror bir o‘zgaruvchidan ko‘rsatkichni olish operatori yoki nol dastlabki qiymat bo‘lishi mumkin. Turlashtirilmagan (tipiklashtirilmagan) ko‘rsatkichlar ko‘rsatkich turlarini qayta o‘zgar-

tirishni ko‘zda tutgan dasturlarda qo‘llanadi hamda void turdagi o‘zgaruvchiga ko‘rsatkich sifatida taqdim etiladi.

Biror bir o‘zgaruvchi manzilining qiymatiga ega bo‘lish hamda uni ko‘rsatkichga berish uchun «&» operatori qo‘llanadi.

Misol:

```
int I=100;
int*p=&I;
```

«*» — teskari operator bo‘lib, ko‘rsatkichda manzili saqlanayotgan uya qiymatiga murojaat qilish imkonini beradi.

Misol:

```
int I=100;
int J=0;
int*p=&I;
J=(p+sizeof(int));
```

Ko‘rsatkichlar ustida o‘tkaziladigan operatsiyalar

Ko‘rsatkichlar ustida unar operatsiyalar bajarish mumkin: inkrement va dekrement. ++ va —— operatsiyalarini bajarishda ko‘rsatkich qiymati ham qo‘llanayotgan ko‘rsatkich murojaat qilgan tur uzunligiga ko‘payadi yoki kamayadi.

Misol:

```
int*ptr, a[10];
ptr=&a[5];
ptr++;      /* = a[6] elementining adresiga*/
ptr——;   /* = a[5] elementining adresiga*/
```

Qo‘shish va ayirish binar operatsiyalarida ko‘rsatkich va int turining qiymati ishtirok etishi mumkin. Bunda operatsiya natijasida dastlabki turning ko‘rsatkichi yuzaga keladi, uning qiymati esa dastlabkisidan ko‘rsatilgan elementlar soniga ko‘proq yoki kamroq bo‘ladi.

Misol:

```
int*ptr1, *ptr2, a[10];
int i=2;
ptr1=a+(i+4);      /* = a[6] elementining adresiga*/
ptr2=ptr1-i;       /* = a[4] elementining adresiga*/
```

Ayirish operatsiyasida bitta turga mansub bo‘lgan ikkita ko‘rsatkich ishtirok etishi mumkin. int turi mana shunday operatsiya natijasiga ega bo‘ladi hamda kamayuvchi va ayiruvchi o‘rtasidagi dastlabki tur

elementlarining soniga teng, bundan tashqari agar birinchi adres kichikroq (to‘g‘rirog‘i, yoshroq) bo‘lsa, u holda natija manfiy qiymatga ega bo‘ladi.

Misol:

```
int *ptr1, *ptr2, a[10];
int i;
ptr1=a+4;
ptr2=a+9;
i=ptr1—ptr2; /*=5 */
i=ptr1—ptr2; /*=—5 */
```

Bir turga taalluqli bo‘lgan ikkita ko‘rsatkich qiymatlarini ==, !=, <, <=, >, >= operatsiyada o‘zaro qiyoslash mumkin. Bunda ko‘rsatkichlarning qiymatlari shunchaki butun sonlar sifatida olib qaraladi, qiyoslash natijasi esa 0 (yolg‘on) yoki 1 (haqiqat) ga teng bo‘ladi.

Misol:

```
int *ptr1, *ptr2, a[10];
ptr1=a+5;
ptr2=a+7;
if(ptr1>ptr2) a[3]=4;
```

Bu misolda ptr1 ning qiymati ptr2 ning qiymatidan kamroq, shuning uchun a[3]=4 operatori bajarilmay qoladi.

Iqtibos tushunchasi

Iqtibos (ssilka) — iqtibosni nomlantirishda ko‘rsatilgan obyekt nomining sinonimi.

Ko‘rsatkichni e’lon qilish formati

```
tur & ism =ism_obyekt;
```

Misollar:

```
int x; // o‘zgaruvchini aniqlash
int& sx=x; //x o‘zgaruvchiga iqtibosni aniqlash
const char & CR=’\n’; //konstantaga iqtibosni aniqlash
```

Iqtiboslar bilan ishlash qoidalari

1) O‘zgaruvchi iqtibos, agar u funksiya parametri bo‘lmasa, extern sifatida tavsiflanmagan bo‘lsa yoki sinf maydoniga iqtibos qilmasa, o‘ziga tavsif berilayotganda ochiq-oydin nomlanishi kerak.

2) Nomlangandan so‘ng, iqtibosga boshqa qiymat berilishi mumkin emas.

3) Iqtiboslarga ko‘rsatkichlar, iqtiboslar massivlari va iqtiboslarga iqtiboslar bo‘lishi mumkin emas.

4) Iqtibos ustida o‘tkazilgan operatsiya o‘zi iqtibos qilayotgan qiymatning o‘zgarishiga olib keladi.

Iqtibos xotirada qo‘shimcha maydonni egallamaydi, u shunchaki obyektning boshqa nomi bo‘ladi, xolos.

Misol:

```
#include<iostream.h>
void main()
{
int I=123;
int &si=I;
cout<<"\ni="<<I<<" si="<<si;
I=456;
cout<<"\ni="<<I<<" si="<<si;
I=0; cout<<"\ni="<<I<<" si="<<si;
}
```

Kelib chiqadi:

```
I-123 si=123
I=456 si=456
I=0 si=0
```

2.3. Boshqaruvchi tuzilmalar

Ma’lumotlarni kiritish va chiqarish operatorlari

C++ tilida qurilma kiritish va chiqarish vositalari yo‘q. Bu ish standart kutubxonalarda saqlanadigan funksiyalar, turlar va obyektlar yordamida amalga oshiriladi. C++ sinflari kutubxonalaridan foydalanilganda, iostream.h kutubxona fayli qo‘llanib, unda cin klaviaturasidan kiritiladigan ma’lumotlarning hamda cout displeyi ekraniga chiqariladigan ma’lumotlarning standart oqimlari, shuningdek, tegishli operatsiyalar belgilangan. Bu operatsiyalar quyidagicha:

1) << — ma’lumotlarni oqimga yozish operatsiyasi;

2) >> — ma’lumotlarni oqimdan o‘qish operatsiyasi.

Masalan:

```
#include<iostream.h>;
.....
cout<<"\nElementlar miqdorini kiriting:";
cin>>n;
```

Cout dan sonlarni chiqarishda foydalanish

```
cout<<1001;
cout<<0.12345;
```

Bir paytning o‘zida bir nechta qiymatni chiqarish

```
cout<<1<<0<<0<<1;
cout<< ″Mening yaxshi ko‘rgan sonim teng″ <<1001;
cout<< ″Bu″ <<20<< ″yil ichida mening oylik maoshim edi″
<<493.34<<endl;
```

Agar kursorni keyingi satr boshiga surish kerak bo‘lsa, chiqarish oqimiga yangi satr belgisi (\n) ni joylashtirish mumkin.

```
cout<< ″Bu birinchi satr\nBu ikkinchi satr″;
cout<<1<<′\n′<<0<<′\n′<<0<<′\n′<<1;
```

Dasturingiz nimaga mo‘ljallanganiga qarab, ehtimol, sonlarni sakkizlik yoki o‘n oltilik ko‘rinishda chiqarish talab qilinib qolar. Buning uchun chiqarish oqimining ichiga dec, oct, hex modifikatorlarini joylashtirish mumkin.

```
cout <<″Sakkizlik:″<<oct<< 10 <<′   ′<< 20 << endl;
cout <<″O‘n oltilik:″<<hex<<10<<′   ′<< 20 << endl;
cout <<″O‘nlik:″<<dec << 10 <<′     ′<< 20 << endl;
```

Standart qurilmaga xatolarni chiqarish

O‘zingizga ma’lum bo‘lganidek, cout dan foydalanib, operatsiya tizimining chiqarishini qayta tayinlash operatorlari yordamida siz dasturni qurilmaga yoki faylga qayta yo‘naltirishingiz mumkin. Biroq, agar sizning dasturlaringiz xatoga duch kelib qolsa, siz xato haqidagi xabar ekrandan qayta yo‘naltirilishini, ehtimol, xohlamassiz. Xatolar haqidagi xabarlar faylga qayta yo‘naltirilsa, xato paydo bo‘lganligi haqida foydalanuvchi bexabar qolishi mumkin.

Agar sizning dasturingiz xato haqidagi xabarni chiqarib berishi kerak bo‘lsa, siz chiqarish oqimi cerr dan foydalanishingiz kerak. C++cerr ni operatsiya tizimining xatolar standart qurilmasi bilan bog‘laydi. Keyingi CERR.CPP dasturi chiqarish oqimi cerr dan ekranga «Bu xabar hammavaqt paydo bo‘ladi» xabarini chiqarish uchun foydalanadi:

```
#include<iostream.h>
void main (void)
{
```

```
cerr<<″Bu xabar hammavaqt paydo bo‘ladi″;
}
```

Ushbu dasturni ko‘chiring va ishga tushiring. Keyin chiqarishni qayta tayinlash operatoridan foydalanib, dastur chiqarishini faylga qayta yo‘naltirishga harakat qilib ko‘ring:

C:\> CERR FALENAME.EXT <ENTER>

Operatsiya tizimi sizning dasturlaringizga standart xatolar qurilmasiga yozilayotgan chiqarishni qayta yo‘naltirishga yo‘l qo‘ymagani uchun xabar sizning ekraningizda paydo bo‘ladi.

Chiqarish kengligini boshqarish

Dastlabki bir nechta dastur sonlarni ekranga chiqarar edi. Bu sonlarning to‘g‘ri aks ettirilishini kafolatlash uchun dasturlar sonlardan oldin va keyin bo‘sh joylar qoldirar edi. Cout yoki cerr ni chiqarishda sizning dasturlaringiz setw (kenglikni o‘rnatish) modifikatoridan foydalanib, har bir son chiqarilishining kengligini ko‘rsatishi mumkin. Setw dan foydalanib dasturlar son egallagan eng kam miqdordagi belgilarni ko‘rsatadi. Masalan, SETW.CPP dasturi 1001 soni uchun 3, 4, 5 va 6 kengliklarini tanlash maqsadida setw modifikatoridan foydalanadi. Setw modifikatoridan foydalanish uchun sizning dasturingiz sarlavha fayli iomanip.h ga ega bo‘lishi kerak:

```
#include<iostream.h>
#include<iomanip.h>
void main (void)
{
cout <<″Mening yaxshi ko‘rgan sonim teng″<< setw(3) <<1001
<< endl;
cout <<″Mening yaxshi ko‘rgan sonim teng″<< setw(3) <<1001
<< endl;
cout <<″Mening yaxshi ko‘rgan sonim teng″<< setw(3) <<1001
<< endl;
cout <<″Mening yaxshi ko‘rgan sonim teng″<< setw(3) <<1001
<< endl;
}
```

Agar kenglikni setw yordamida ko‘rsatayotgan bo‘lsangiz, bu holda siz son egallab turgan belgilar pozitsiyasining minimal miqdorini ko‘rsatgan bo‘lasiz. Avvalgi dasturda setw(3) modifikatori kamida uchta belgini ko‘rsatib turgan edi. Biroq 1001 soni uchtadan ko‘proq belgini talab qilgani uchun cout haqiqiy miqdordan foydalandiki,

bu miqdor ushbu misolda to‘rtga teng edi. Shuni ham ta’kidlash kerakki, kenglikni tanlash uchun setw dan foydalanishda, ko‘rsatilgan kenglik faqat bitta sonni chiqarish uchun mo‘ljallangan. Agar siz bir nechta son uchun kenglikni ko‘rsatmoqchi bo‘lsangiz, bu holda setw dan bir necha marta foydalanishingizga to‘g‘ri keladi.

Shartli operator

Dasturda shoxlanishlarni tashkil qilish uchun, ya’ni ayrim faktorlarga bog‘liq holda turli xatti-harakatlar bajarilishi uchun if operatori qo‘llanadi.

Operator quyidagi formatga ega:

if (ifoda){operator-1;}[else{operator-2;}]

if operatorining bajarilishi ifodani hisoblashdan boshlanadi. Keyin bajarilish quyidagi sxema bo‘yicha amalga oshiriladi:

- agar ifoda haqiqiy bo‘lsa (ya’ni noldan boshqa), bu holda operator-1 bajariladi.
- agar ifoda soxta bo‘lsa (ya’ni 0 ga teng), bu holda operator-2 bajariladi.
- agar ifoda soxta bo‘lsa-yu, operator-2 bo‘lmasa (shart bo‘lmagan konstruksiya kvadrat qavslar ichiga olingan), bu holda if dan keyin turgan operator bajariladi.

Misol:

```
(i<j)
{
        i++;
}
else
{
        j=i-3;
          i++;
}
```

Bu misoldan yana shu narsa ma’limki, operator-1 o‘rnida, xuddi operator-2 o‘rnida bo‘lganidek, murakkab konstruksiyalar bo‘lishi mumkin.

Solib qo‘yilgan if operatorlaridan foydalanishga ham yo‘l qo‘yiladi. if operatori if konstruksiyasi tarkibiga yoki boshqa if operatorining else konstruksiyasiga kiritilgan bo‘lishi ham mumkin.

Misollar:

```
int t=2;
int b=7;
```

```
int r=3;
if(t>b)
{
        if(b<r)
        {
              r=b;
         }
}
else
{
            r=t;
          }
```

Ushbu dasturning bajarilishi natijasida r=2 bo‘ladi.

Davriy tuzilmalarga kirish

Ko‘plab tarkorlanuvchi elementlarga ega bo‘lgan algoritmga mos dasturiy kodni yaratish uchun quyidagi operatorlar tomonidan yaratiladigan davriy (siklik) tuzilmalardan foydalanish kerak.

For operatori

For operatori — davr (sikl)ni tashkil qilishning eng umumiy usuli. U quyidagi formatga ega:

for (1-ifoda; 2-ifoda; 3-ifoda) {tana}

1-ifoda odatda davrni boshqarayotgan o‘zgaruvchilarning dastlabki qiymatini belgilash uchun qo‘llanadi. 2-ifoda davr tanasi bajarilishi mumkin bo‘lgan shartni belgilab beruvchi ifoda. 3-ifoda sikl tanasi har gal bajarilganidan so‘ng siklni boshqaradigan o‘zgaruvchilarning o‘zgarishini belgilaydi.

For operatorini bajarish sxemasi:

1. 1-ifoda hisoblanadi.
2. 2-ifoda hisoblanadi.
3. Agar 2-ifodaning qiymati 0 dan farqli (haqiqiy) bo‘lsa, davr tanasi bajariladi, 3-ifoda hisoblanadi va 2-bandga o‘tish amalga oshiriladi, agar 2-ifoda 0 ga teng (yolg‘on) bo‘lsa, u holda boshqaruv for operatoridan keyin kelgan operatorga uzatiladi.

Shunisi muhimki, shart hammavaqt davr boshida tekshirib olinadi. Bu demak, agar bajarilish sharti avvaldan soxta bo‘lsa, u holda davr tanasi biron marta ham bajarilmasligi mumkin.

Misol:

```
int i, b;
```

```
for (i=1; i<10; i++)
{
        b=i*i;
}
```

Bu misolda 1 dan 9 gacha bo‘lgan sonlar kvadratlari hisoblanadi.

For operatorining ayrim variantlaridan foydalanganda, davrni boshqarayotgan bir nechta o‘zgaruvchilardan foydalanish mumkinligi hisobiga uning moslashuvchanligi oshadi.

Misol:

```
int top, bot;
char string[100], temp;
for (top=0, bot=100; top<bot; top++, bot——)
{
        temp=string[top];
        string[bot]=temp;
}
```

Belgilar satrini aks tartibda yozishni joriy qiladigan ushbu misolda davrni boshqarish uchun ikkita top, bot o‘zgaruvchilari qo‘llanadi. Shuni ham ta’kidlab o‘tamizki, 1- va 3-ifodaning o‘rniga bu yerda vergul bilan ajratilgan va ketma-ket bajariladigan bir nechta ifoda qo‘llanadi.

For operatoridan foydalanishning boshqa bir varianti bu cheksiz davrdir. Bunday davrni tashkil qilish uchun bo‘sh shartli ifodadan foydalanish mumkin, davrdan chiqish uchun esa odatda qo‘shimcha shartdan hamda break operatoridan (biz uni keyinroq ko‘rib chiqamiz) foydalaniladi.

Misol:

```
for(;;)
{
        ...
        ... break;
        ...
}
```

C tilining sintaksisiga muvofiq operator bo‘sh bo‘lishi mumkinligi tufayli, for operatorining tanasi ham bo‘sh bo‘lishi mumkin. Operatorning bunday shakli qidirishni tashkil qilishda qo‘llanishi mumkin.

Misol:

for(i=0; t[i]<10; i++);

Ushbu misolda i davrining o‘zgaruvchisi t massivi birinchi elementi raqamining qiymatini oladi. Bu qiymat 10 dan oshiq bo‘ladi.

While operatori

While davri operatori shartdan avvalgi davr deb ataladi va quyidagi formatga ega bo‘ladi:

while (ifoda) {tana};

Ifoda sifatida C tilining har qanday ifodasidan foydalanishga, tana sifatida esa har qanday, shu jumladan, bo‘sh yoki tarkibli operatordan foydalanishga yo‘l qo‘yiladi. While operatorini bajarish quyidagicha:

1. Ifoda hisoblanadi.

2. Agar ifoda soxta bo‘lsa, u holda while operatorining bajarilishi tugallanadi hamda navbatdagi keyingi operator bajariladi. Agar ifoda haqiqiy bo‘lsa, u holda while operatorining tanasi bajariladi.

3. Jarayon 1-banddan boshlab qaytariladi.

Tur davrining operatori — for (1-ifoda; 2-ifoda; 3-ifoda) {tana} — while operatori bilan quyidagicha almashtirilishi mumkin:

```
1-ifoda;
while (2-ifoda)
{
        tana
        3-ifoda;
}
```

Operator for ni bajarishda bo‘lganidek, while operatorida ham avval shart tekshiriladi. Shuning uchun operator tanasini bajarish hammavaqt ham talab qilinmaydigan vaziyatlarda while operatoridan foydalanish qulay.

For va while operatorlari ichida lokal o‘zgaruvchilarni qo‘llash mumkin bo‘lib, ularni e’lon qilishda bir paytning o‘zida tegishli turlar belgilanishi kerak.

Do while operatori

Do while davri operatori shart belgilangandan keyingi davr operatori deb ataladi hamda davr tanasini aqalli bir marta bajarish lozim bo‘lgan holatlarda qo‘llanadi. Operator formati quyidagi ko‘rinishga ega:

do{tana} while (i);

Do while operatorini bajarish sxemasi:

1. Davr tanasi bajariladi (u tarkibli operator bo‘lishi mumkin).

2. Ifoda hisoblanadi.

3. Agar ifoda soxta bo‘lsa, bu holda do while operatorining bajarilishi tugallanadi va navbatda turgan keyingi operator bajariladi. Agar ifoda haqiqiy bo‘lsa, operatorning bajarilishi 1-banddan davom etadi.

While va do while operatorlari ichiga qo‘yilgan bo‘lishi ham mumkin.

Misol:

```
int i,j,k;
...
i=0; j=0; k=0;
do
{
        i++;
        j——;
        while (a[k]<i)
        {
                k++;
         }
}
while (i<30 && j<-30);
```

Break operatori

Break operatori uni birlashtiruvchi switch, do, for, while davrlardan eng ichkarisida joylashganining bajarilishini tugallash uchun qo‘l-lanadi. Break operatorining bajarilishi tugallangach, boshqaruv tugallangan davrdan keyin turgan operatorga uzatiladi. Shu yo‘l bilan muddatdan oldin davrdan chiqish ta’minlanadi.

Continue operatori

Continue operatori, break operatori kabi, faqat davr operatorlari ichida qo‘llanadi, biroq, undan farqli o‘laroq, dasturning bajarilishi uzilgan davrdan keyin turgan operatordan emas, balki uzilgan davr boshidan davom ettiriladi.

Misol:

```
int a,b;
for (a=1, b=0; a<100; b+=a, a++)
```

```
{
        if (b%2 !=0) continue;
        ...
/*juft summalarga ishlov berish*/
}
```

Ushbu misolda ko‘p nuqta bilan ifodalangan xatti-harakatlar faqat toq b lardagina bajariladi, chunki 1 dan a gacha bo‘lgan sonlar summasi toq bo‘lib, qolgani uchun continue operatori boshqaruvni for davrining navbatdagi o‘ramiga uzatadi hamda bunda ishlov berish operatorlarini bajarmaydi. Continue operatori, break operatori kabi, o‘zini o‘rab turgan davrlarning eng ichkaridagisini uzib qo‘yadi.

Kalit bo‘yicha o‘tish operatori

Kalit bo‘yicha o‘tish — switch operatori umumiy ko‘rinishi quyidagicha:

```
Switch(<ifoda>)  {
Case <1-qiymat>:<1-operator>
                    ...
 break;
                    ...
 default: <operator>
                    ...
  case: <n-operator>;
   }
```

Oldin qavs ichidagi butun ifoda hisoblanadi va uning qiymati hamma variantlar bilan solishtiriladi. Biror variantga qiymat mos kelsa, shu variantda ko‘rsatilgan operator bajariladi. Agar biror variant mos kelmasa, default orqali ko‘rsatilgan operator bajariladi. Break operatori ishlatilmasa shartga mos kelgan variantdan tashqari keyingi variantdagi operatorlar ham avtomatik bajariladi. Default; break va belgilangan variantlar ixtiyoriy tartibda kelishi mumkin. Default yoki break operatorlarini ishlatish shart emas. Belgilangan operatorlar bo‘sh bo‘lishi ham mumkin. Misol tariqasida bahoni son miqdoriga qarab aniqlash dasturini ko‘ramiz:

```
#include <iostream.h>
int baho;
cin>> baho;
```

```
switch(baho)
{case 2:cout <<″ \n yomon″;break;
 case 3:cout <<″ \n  o‘rta″;break;
 case 4:cout <<″ \n  yaxshi″;break;
 case 5:cout <<″ \n  a’lo″;break;
 default:cout <<″ \n baho noto‘g‘ri kiritilgan″;
}
}
```

Keyingi misolimizda kiritilgan simvol unli harf ekanligi aniqlanadi:

```
#include <iostream.h>
int baho; char c; cin >> c;
switch(c)
{case ′ a ′:
case ′ u ′:
case ′ o ′:
case ′ i ′:
cout <<″\n Kiritilgan simvol unli harf ″;break;
default: cout <<″ \n Kiritilgan simvol unli harf emas ″;
}
}
```

2.4. C++da funksiyalar

Dastur hajmining ko‘payishi bilan uning xotirasida hamma detallarni saqlab turish imkoni qiyinlashadi. Dasturni soddalashtirish uchun u qismlarga bo‘linadi. C++ da masala funksiyalar yordamida soddaroq masalachalarga bo‘linishi mumkin. Shuningdek, masalaning funksiyalarga bo‘linishi kodning ortiqchaliligini bartaraf etish imkonini ham beradi, chunki funksiya bir marta yoziladi, ko‘p marta chaqiriladi. Tarkibida funksiya bo‘lgan dasturni sozlash oson bo‘ladi.

Ko‘pincha qo‘llanayotgan funksiyalarni kutubxonalarga joylashtirish mumkin. Shunday qilib, sozlashda va kuzatib borishda ancha sodda dasturlar yaratiladi.

Funksiyalarni e’lon qilish va aniqlash

Funksiya — bu tavsiflar va operatorlarning nomlangan ketmaketligi bo‘lib, tugallangan xatti-harakatlarni, masalan, massivni shakllantirish, massivni bosib chiqarish va h.k. larni bajaradi.

Funksiya, birinchidan, C++ ning hosila turlaridan biri, ikkinchidan esa, minimal bajarilayotgan dastur moduli hisoblanadi.

Har qanday funksiya e'lon qilinishi va aniqlanishi kerak.

Funksiyani e'lon qilishda (prototip, sarlavha) unga nom, qaytarilayotgan qiymat turi va uzatilayotgan parametrlar ro'yxati beriladi.

Funksiyaning aniqlanishi, e'londan tashqari, yana tavsiflar va operatorlar ketma-ketligidan iborat funksiya tanasini bildiradi.

Funksiyaning__tur nomi([formal__parametrlar__ro'yxati])

(funksiya__tanasi)

Funksiya__ tanasi bu — blok yoki tarkibli operatordir. Funksiya ichida boshqa funksiyani aniqlash mumkin emas. Funksiya tanasida funksiyaning olingan qiymatini chaqirilish nuqtasiga qaytaradigan operator bo'lishi lozim. U ikkita shaklga ega bo'ladi:

1) return ifoda;

2) return.

Birinchi shakl natijani qaytarish uchun qo'llanadi, shuning uchun aniqlashdagi funksiya qanday turga ega bo'lsa, ifoda ham shunday turga ega bo'lishi kerak. Agar funksiya qiymatni qaytarmasa, ya'ni tur void ga ega bo'lsa, ikkinchi shakl qo'llanadi. Dasturchining o'zi bu operatorni funksiya tanasida qo'llamasligi mumkin, kompilyator uni funksiya oxiriga avtomatik tarzda qo'shib qo'yadi.

Qaytarilayotgan turning qiymati, massiv va funksiyadan tashqari, har qanday turdagi qiymat bo'lishi mumkin, ammo massiv yoki funksiyaga ko'rsatkich ham bo'lishi mumkin.

Formal parametrlar ro'yxati — bu funksiyaga uzatilishi lozim bo'lgan qiymatlar. Ro'yxat elementlari vergullar bilan ajratiladi. Har bir parametr uchun tur va nom ko'rsatiladi. E'londa nomlarni ko'rsatmasa ham bo'ladi.

Funksiya tanasida yozilgan operatorlar bajarilishi uchun funksiyani chaqirib olish lozim. Chaqirishda funksiyaning nomi va faktik parametrlari ko'rsatiladi. Funksiya tanasi operatorlarini bajarishda faktik parametrlar formal parametrlarning o'rnini egallaydi. Faktik va formal parametrlar miqdori va turiga ko'ra bir-biriga mos kelishi kerak.

Kompilyator chaqirilishning to'g'riligini tekshirish imkoniga ega bo'lishi uchun funksiyani e'lon qilish funksiya chaqirilishdan oldin matnda bo'lmog'i lozim. Agar funksiya void bo'lmagan turga ega bo'lsa, u holda uning chaqirilishi ifodaning operatsiya bajarilayotgan elementi bo'lishi mumkin.

Misol:

Aytaylik, uchburchak tomonlarining koordinatalari berilgan. Agar shunday uchburchak mavjud bo‘lsa, uning maydoni topilsin.

I. Matematik model:

1) I=sqtr(pow(x1-x2,2)+pow(y1-y2,2));//uchburchak tomonining uzunligi;

2) p=(a+b+c)/2;

s=sqrt(p*(p-a)*(p-b)*(p-c));//Geron formulasi;

3) uchburchak mavjudligini tekshirish

(a+b>c&&a+c>b&&c+b>a)

II. Algoritm:

1) (x1,u1), (ch2,u2), (ch3,u3) uchburchagi tomonlarining koordinatlari kiritilsin;

2) ab, bc, ca tomonlarining uzunligi hisoblansin;

3) shunday tomonlarga ega bo‘lgan uchburchakning mavjudligi tekshirilsin. Agar mavjud bo‘lsa, unda uning maydoni hisoblansin va natijasi chiqarilsin;

4) agar mavjud bo‘lmasa, xabar chiqarilsin;

5) agar hamma koordinatlar 0 ga teng bo‘lsa, unda tamom, aks holda 1-bandga qaytiladi.

```
#include<iostream.h>
#include<math.h>
double line(double x1, double y1, double x2, double y2)
{
//Funksiya x1,y1 x2, y2 koordinatalariga ega bo‘lgan kesim uzunligini qaytarib beradi:
return sqrt(pow(x1-x2,2)+pow(y1-y2,2));
}
double square(double a, double b, double c);

{
//funksiya a, b, c uzunlikdagi tomonlarga ega bo‘lgan uchburchak maydonini qaytarib beradi.
double s, r=(a+b+c)/2;
return s=sqrt(p*(p-a)*(p-b)*(p-c));//Geron formulasi
}
bool triangle(double a, double b, double c);
```

```
{
//agar uchburchak mavjud bo‘lsa, true ni qaytarib beradi;
if(a+b>&&a+c>b&&c+b>a)return true;
else return false;
}
void main()
{
double x1=1,y1,x2,y2,x3,y3;
double point1_2,point1_3,point2_3;
do
{
cout<<"\n Uchburchak koordinatalari:";
cin>>x1>>y1>>x2>>y2>>x3>>y3;
point1_2=line(x1,y1,x2,y2);
point1_3=line(x1,y1,x3,y3);
point2_3=line(x2,y2,x3,y3);
if(triangle(point1_2,point1_3,point2_3)==true)
cout<<"S="<<square(point1_2,point2_3,point1_3)<<'\n';
else cout<<"\n Uchburchak mavjud emas";
}
while(!(x1==0&&y1==0&&x2==0&&y2==0&&x3==0&&y3==0));
}
```

Funksiyalar prototiplari

Funksiyaga murojaat qilish mumkin bo‘lsin uchun xuddi shu faylning o‘zida funksiya aniqlovchisi yoki tavsifi (prototipi) bo‘lmog‘i lozim.

```
double line(double x1, double y1, double x2 double y2);
double square(double a, double b, double c);
double triangle(double a, double b, double c);
double line(double, double, double, double);
double square(double, double, double);
double triangle(double, double, double).
```

Bu yuqorida tavsiflari keltirilgan funksiyalarning prototiplaridir.

Prototiplar bo‘lganda, chaqirilayotgan funksiyalar chaqirayotgan funksiyalar bilan bitta faylda bo‘lishlari shart emas, balki ular alohida modullar ko‘rinishida rasmiylashtirilishi hamda ko‘chirilgan holda obyektlar modullari kutubxonasida saqlanishlari mumkin. Xuddi shu narsa standart modullardagi funksiyalarga ham tegishli. Bu holda obyekt modullari sifatida translatsiya qilinib, rasmiylashtirilib bo‘lingan

kutubxona funksiyalarining aniqlovchilari kompilyator kutubxonasida bo‘ladi, funksiyalar tavsiflarini esa dasturga qo‘shimcha ravishda kiritish lozim bo‘ladi. Bu ish include<fayl nomi> protsessor buyruqlari yordamida amalga oshiriladi.

Fayl_nomi sarlavhaviy faylni aniqlaydi. Sarlavhaviy fayl esa berilgan funksiyalar kompilyatori uchun standart bo‘lgan guruhlar prototipiga ega bo‘ladi. Masalan, deyarli barcha dasturlarda biz kiritish-chiqarish obyektlar oqimining tavsifi uchun #include <iostream.h> buyrug‘idan hamda ularga mos operatsiyalardan foydalandik.

Katta miqdordagi funksiyalardan iborat bo‘lgan hamda turli modullarda joylashtirilgan dasturlarni ishlab chiqishda funksiyalar prototiplari va tashqi obyektlarning tavsiflari (konstantalar, o‘zgaruvchilar, massivlar) alohida faylga joylashtiriladi. Bu fayl esa include ″fayl_nomi″ direktivasi yordamida har bir modulning boshiga kiritiladi.

Funksiya parametrlari

Chaqirilayotgan va chaqirayotgan funksiyalar o‘rtasida axborot almashinishning asosiy usuli bu parametrlardir. Parametrlarni funksiyaga uzatishning ikkita usuli mavjud: manzil bo‘yicha va qiymati bo‘yicha.

*Qiymati bo‘yicha uzatish*da quyidagi xatti-harakatlar bajariladi:

a) faktik parametrlar o‘rnida turgan ifodalar qiymatlari hisoblanadi;

b) funksiyaning formal parametrlari uchun stekda xotira ajratiladi;

d) har bir faktik parametrga formal parametr qiymati beriladi, bunda turlarning o‘zaro muvofiqligi tekshiriladi hamda zarurat tug‘ilganda ular qayta o‘zgartiriladi.

Misol:

```
double square(double a, double b, double c);
{
//funksiya a, b, c uzunlikdagi tomonlarga ega bo‘lgan uchburchak maydonini qaytarib beradi.
double s, r=(a+b+c)/2;
return s=sqrt(p*(p-a)*(p-b)*(p-c));//Geron formulasi
}
```

Shunday qilib, stekka faktik parametrlarning nusxalari kiritiladi va funksiya operatorlari ushbu nusxalar bilan ish olib boradi. Faktik

parametrlarning o‘ziga funksiyaning kirish huquqi yo‘q, demak, ularni o‘zgartirish imkoni ham yo‘q.

*Manzil bo‘yicha uzatish*da stekka parametrlar manzillarining nusxalari kiritiladi, demakki, funksiyada faktik parametr joylashtirilgan xotira uyasiga kirish huquqi paydo bo‘ladi va funksiya bu parametrni o‘zgartirishi mumkin.

```
void change(int*a,int*b)//manzil bo‘yicha uzatish;
{int r=*a; *a=*b;*b=r;}
int x=1,y=5;
change(&x,&y);
cout<<″x=″<<x<<″y=″<<y;
x=5y=1 kelib chiqadi.
```

Manzil bo‘yicha uzatish uchun iqtiboslar ham qo‘llanishi mumkin. Iqtibos bo‘yicha uzatishda funksiyaga chaqirish paytida ko‘rsatilgan parametr manzili uzatiladi, funksiya ichida esa parametrga barcha murojaatlarning sezilmagan holda nomlari bekor qilinadi:

```
void change(int &a,int &b)
{int r=a; a=b;b=r;}
int x=1,y=5;
change(x,y);
cout<<″x=″<<x<<″y=″<<y;
x=5y=1 kelib chiqadi.
```

Ko‘rsatkichlar o‘rniga iqtiboslardan foydalanish dasturning o‘qilishini yaxshilaydi, chunki bu o‘rinda nomni bekor qilish operatsiyasini qo‘llamasa ham bo‘ladi. Qiymat bo‘yicha uzatish o‘rniga iqtiboslardan foydalanish samaraliroq hamdir, chunki parametrlarni nusxalashni talab qilmaydi. Agar funksiya ichida parametrning o‘zgarishini taqiqlash lozim bo‘lib qolsa, bu holda const modifikatori qo‘llanadi. Const modifikatorini funksiyada o‘zgarishi ko‘zda tutilmagan barcha parametrlar oldidan qo‘yish tavsiya qilinadi (undagi qaysi parametrlar o‘zgaradi-yu, qaysilari o‘zgarmasligi sarlavhadan ko‘rinib turadi).

Lokal va global o‘zgaruvchilar

Berilgan funksiya ichida qo‘llanadigan o‘zgaruvchilar lokal deb ataladi. Ular uchun stekda xotira ajratilmaydi, shuning uchun, ish

tugagach, funksiyalar xotiradan chiqarib tashlanmaydi. Ko‘rsatkichni lokal o‘zgaruvchiga qaytarish mumkin emas, chunki bunday o‘zgaruvchi ajratib bergan xotira bo‘shatila boshlaydi:

```
int*f()
{
int a;
....
return&a;//NOTO‘G‘RI
}
```

Global o‘zgaruvchilar — bu funksiyadan tashqarida tavsiflangan funksiyalar. Ular shunday nomli lokal funksiyalar bo‘lmagan barcha funksiyalarda ko‘rinadi.

```
Misol:
int a,b;//global o‘zgaruvchilar
void change()
{
int r;//lokal o‘zgaruvchi
r=a;a=b;b=r;
}
void main()
{
cin>>a>>b;
change();
cout<<″a=″<<a<<″b=″<<b;
}
```

Funksiyalar o‘rtasida ma’lumotlarni uzatish uchun global o‘zgaruvchilardan ham foydalanish mumkin, lekin bunday qilish tavsiya etilmaydi, chunki bu dasturni sozlashni qiyinlashtiradi hamda funksiyalarni kutubxonaga joylashga to‘sqinlik qiladi. Funksiyalar maksimal mustaqil bo‘lishiga, funksiya prototipi esa ularning interfeysini to‘lig‘icha aniqlashiga intilish kerak.

Dastlabki (yashirilgan) parametrlar qiymatiga ega bo‘lgan funksiyalar

Funksiyani aniqlashda dastlabki (yashirilgan) parametr qiymati bo‘lishi mumkin. Agar funksiyani chaqirishda tegishli parametr tushirib qoldirilgan bo‘lsa, mana shu qiymat qo‘llanadi. Bunday parametrning o‘ng tomonida tavsiflangan parametrlar ham yashirilgan bo‘lishi lozim.

Misol:
void print(int value=1)
{cout<<'\n'<<"Uy raqami:"<<value:}

Chaqirishlar:
1. print();
   Xulosa: Uy raqami: 1
2. print(15);
   Xulosa: Uy raqami: 15

Taqdim etiladigan (inline) funksiyalar

C++ dagi ayrim funksiyalarni inline rasmiy so‘zini qo‘llagan holda aniqlash mumkin. Bunday funksiya taqdim etilayotgan yoki o‘rnatilayotgan funksiya deb ataladi.

Masalan:

inline float line(float x1, float y1, float x2=0, float y2=0)

{return sqrt(pow(x1-x2)+pow(y1-y2,2));}// funksiya (x1,u1) koordinatali nuqtadan (x2,u2) koordinatali nuqtagacha bo‘lgan masofani orqaga qaytaradi.

Qo‘yilgan funksiyaning har bir chaqirishiga ishlov berar ekan, kompilyator dastur matniga dastur tanasi operatorlari kodini joylashtirishga urinadi. Inline spetsifikatori funksiya uchun ichki bog‘lashni aniqlaydi. Ichki bog‘lash shundan iboratki, bunda kompilyator funksiyani chaqirish o‘rniga funksiya kodining buyruqlarini qo‘yadi. Bunda dastur hajmi kattalashishi mumkin, ammo chaqirilayotgan funksiya boshqaruvni uzatish va undan qaytishga ketadigan sarflar bo‘lmaydi. Agar funksiya tanasi bir necha operatorlardan iborat bo‘lsa, o‘rniga o‘rin qo‘yiladigan funksiyalar qo‘llanadi.

Quyidagi funksiyalar o‘rniga o‘rin qo‘yiladigan bo‘la olmaydi:

1. Rekursiv funksiyalar.

2.Chaqirilishi funksiyalarning aniqlanishidan oldin joylashtiriladigan funksiyalar.

3. Ifodada bittadan ortiq marta chaqiriladigan funksiyalar.

4. Davrlar, ulagichlar va uzatish diapazonlariga ega bo‘lgan funksiyalar.

5. O‘rniga o‘rin qo‘yishni amalga oshirish uchun o‘ta katta hajmga ega bo‘lgan funksiyalar.

Funksiyalarni ortiqcha yuklash

Ortiqcha yuklashning maqsadi shundan iboratki, bunda bitta nomga ega bo‘lgan funksiya turlicha bajarilishi kerak hamda unga murojaat

qilinganda har xil turlarga va har xil sondagi faktik parametrlarga ega bo‘lgan har xil qiymatlarni qaytarib berishi kerak. Ortiqcha yuklashni ta’minlash uchun har bir ortiqcha yuklangan funksiya uchun qaytarib berilayotgan qiymatlar va uzatilayotgan parametrlarni aniqlash kerak. Bu ish shunday amalga oshirilishi kerakki, bunda har bir ortiqcha yuklangan funksiya xuddi shu nomli boshqa funksiyadan ajralib tursin. Kompilyator faktik parametrlar turi bo‘yicha qanday funksiyani tanlab olishni aniqlab beradi.

Misol:

```
#include<iostream.h>
#include<string.h>
int max(int a, int b)
{
if(a>b) return a;
else return b;
}
float max(float a, float b)
{
if(a>b) return a;
else return b;
}
void main()
{
int a1,b1;
float a2, b2;
cout<<″\nfor int:\n″;
cout<<″a=?″; cin>>a1;
cout<<″b=?″; cin>>b1;
cout<<″\nMAX=″<<max(a1,b1)<<′\n′;
cout<<″\nfor float:\n″;
cout<<″a=?″; cin>>a2;
cout<<″b=?″; cin>>b2;
cout<<″\nMAX=″<<max(a2,b2)<<′\n′;
```

Ortiqcha yuklangan funksiyalarni tavsiflash qoidalari:

1) Ortiqcha yuklangan funksiyalar bitta ko‘rish sohasida joylashtirilgan bo‘lishi kerak.

2) Ortiqcha yuklangan funksiyalar yashiringan parametrlarga ega bo‘lishi mumkin, bunda turli funksiyalardagi bitta parametrning

qiymatlari o‘zaro mos bo‘lishi kerak. Ortiqcha yuklangan funksiyalarning turli variantlarida turli miqdordagi yashiringan parametrlar bo‘lishi mumkin.

3) Agar funksiyalar parametrlarining tavsifi faqat const modifikatori bilan yoki iqtibosning mavjudligi bilan farqlansa, funksiyalar ortiqcha yuklangan bo‘lolmaydi.

Masalan, int&f1(int&,const int&){...} va int f1(int,int){...} funksiyalari ortiqcha yuklangan emas, chunki funksiyalarning qaysi biri chaqirilayotganini kompilyator bila olmaydi: parametrni qiymat bo‘yicha uzatayotgan hamda parametrni manzil bo‘yicha uzatayotgan funksiyalarning chaqirilishi o‘rtasida sintaktik (ma’no bo‘yicha) farq yo‘q.

2.5. Massivlar bilan ishlash

Massiv tushunchasi

Massiv — bu bitta turga mansub bir nechta o‘zgaruvchilar to‘plami. TYPE turidagi LENGTH ta elementdan iborat a nomli massiv shunday e’lon qilinadi:

```
type a[length];
```

Bu maxsus a[0], a[1], ..., a[length-1] nomlarga ega bo‘lgan type turidagi o‘zgaruvchilarning e’lon qilinishiga to‘g‘ri keladi. Massivning har bir elementi o‘z raqamiga — indeksiga ega. Massivning x-elementiga kirish indekslash operatsiyasi yordamida amalga oshiriladi:

```
int x=...;                  //butun sonli indeks
TYPE value=a[x];            //x-elementni o‘qish
a[x]=value;                 //x-elementga yozish
```

Indeks sifatida butun tur qiymatini chiqarib beradigan har qanday ifoda qo‘llanishi mumkin: char, short, int, long. C da massiv elementlarining indekslari 0 dan boshlanadi (1 dan emas), LENGTH elementdan iborat bo‘lgan massivning oxirgi elementining indeksi esa — bu LENGTH-1 (LENGTH emas). Shuning uchun massivning barcha elementlari bo‘yicha davr — bu:

```
TYPE a[LENGTH]; int indx;
for(indx< LENGTH; indx++)
...a[indx]...;
```

indx< LENGTH ning qiymati indx<= LENGTH-1 qiymatiga teng. Massiv chegarasidan tashqariga chiqish (ya’ni mavjud bo‘lmagan elementni o‘qish-yozishga urinish) dastur xulq-atvorida kutilmagan

natijalarga olib kelishi mumkin. Shuni ta'kidlab o'tish joizki, bu eng ko'p tarqalgan xatolardan biridir.

Statik massivlarni nomlab e'lon qilish mumkin, bunda massivlar elementlarining qiymatlari vergul bilan ajratilgan shakldor qavs {} ichida sanab o'tiladi. Agar massiv uzunligiga qaraganda kamroq element berilgan bo'lsa, qolgan elementlar 0 hisoblanadi:

```
int a10[10]={1, 2, 3, 4};        //va 6 ta nol
```

Agar nomlangan massivning tavsifida uning o'lchamlari ko'rsatilmagan bo'lsa, u kompilyator tomonidan sanab chiqiladi:

```
int a3[]={1, 2, 3};              //go'yo a3[3]
```

Massivlarni navlarga ajratish

Navlarga ajratish — bu berilgan ko'plab obyektlarni biror bir belgilangan tartibda qaytadan guruhlash jarayoni.

Massivlarning navlarga ajratilishi tez harakatlanuvchiligiga ko'ra farqlanadi. Navlarga ajratishning n*n ta qiyoslashni talab qilgan oddiy usuli va n*In(n) ta qiyoslashni talab qilgan tez usuli mavjud. Oddiy usullar navlarga ajratish tamoyillarini tushuntirishda qulay hisoblanadi, chunki sodda va kalta algoritmlarga ega. Murakkablashtirilgan usullar kamroq sonli operatsiyalarni talab qiladi, biroq operatsiyalarning o'zi murakkabroq, shu sababli uncha katta bo'lmagan massivlar uchun oddiy usullar ko'proq samara beradi.

Oddiy usullar uchta asosiy kategoriyaga bo'linadi:

— oddiy kiritish usuli bilan navlarga ajratish;
— oddiy tanlash usuli bilan navlarga ajratish;
— oddiy almashtirish usuli bilan navlarga ajratish.

Oddiy kiritish usuli bilan navlarga ajratish

Massiv elementlari avvaldan tayyor berilgan va dastlabki ketma-ketliklarga bo'linadi. 1=2 dan boshlab, har bir qadamda dastlabki ketma-ketlikdan 1-element chiqarib olinadi hamda tayyor ketma-ketlikning kerakli o'rniga kiritib qo'yiladi. Keyin bittaga ko'payadi va h.k.

44	55	12	42	94	18

Tayyor dastlabki ketma-ketlik

Kerakli joyni izlash jarayonida ko'proq o'ndan bitta pozitsiyadan tanlab olingan elementni uzatish amalga oshiriladi, ya'ni tanlab

olingan element, J:=I-1 dan boshlab, navlarga ajratib bo‘lingan qismning navbatdagi elementi bilan qiyoslanadi. Agar tanlab olingan element a[I] dan katta bo‘lsa, uni navlarga ajratish qismiga qo‘shadilar, aks holda a[J] bitta pozitsiyaga suriladi, tanlab olingan elementni esa navlarga ajratilgan ketma-ketlikning navbatdagi elementi bilan qiyoslaydilar. To‘g‘ri keladigan joyni qidirish jarayoni ikkita turlicha shart bilan tugallanadi:

— agar a[j]>a[i] elementi topilgan bo‘lsa;

— agar tayyor ketma-ketlikning chap uchiga yetilgan bo‘lsa.

```
int i, j, x;
fjr(i=1; i<n; i++)
{
x=[i];// kiritib qo‘yishimiz lozim bo‘lgan elementni esda saqlab qolamiz
j=i-1;
while(x<a[j]&&j>=0)//to‘g‘ri keladigan joyni qidirish
}
a[j+1]=a[j];//o‘ngga surilish
j——;
}
a[j+1]=x;//elementni kiritish
}
```

Oddiy tanlash usuli bilan navlarga ajratish

Massivning minimal elementi tanlanadi hamda massivning birinchi elementi bilan joy almashtiriladi. Keyin jarayon qolgan elementlar bilan takrorlanadi va h.k.

44	55	12	42	94	18

```
int i,min,n, n_ min,j;
for(i=0;i<n—1;i++)
{
min=a[i];n_min=i; //minimal qiymatni qidirish
for(j=i+1;j<n;j++)
                if(a[j]<min){min=a[j];n+min=j;}
                a[n_min]=a[i];//almashtirish
                  a[i]=min;
}
```

Oddiy almashtirish usuli bilan navlarga ajratish

Elementlar juftlari oxirgisidan boshlab qiyoslanadi va o‘rin almashinadi. Natijada massivning eng kichik elementi uning eng chapki elementiga aylanadi. Jarayon massivning qolgan elementlari bilan davom ettiriladi.

44	55	12	42	94	18

```
for(int i=1;i<n;i++)
for(int j=n—1;j>=i;j——)
if(a[j]<a[j—1])
{int r=a[j];a[j]=a[j—1];a[j—1]=r;}
}
```

Ko‘p o‘lchamli massivlar

C++ da massivning eng umumiy tushunchasi — bu ko‘rsatkichdir, bunda har xil turdagi ko‘rsatkich bo‘lishi mumkin, ya’ni massiv har qanday turdagi elementlarga, shu jumladan, massiv bo‘lishi mumkin bo‘lgan ko‘rsatkichlarga ham ega bo‘lishi mumkin. O‘z tarkibida boshqa massivlarga ham ega bo‘lgan massiv ko‘p o‘lchamli hisoblanadi.

Bunday massivlarni e’lon qilishda kompyuter xotirasida bir nechta turli xildagi obyekt yaratiladi. Masalan, int arr[4][3]:

```
Arr
↓
arr[0]  →   arr[0][0]   arr[0][1]   arr[0][2]
arr[1]  →   arr[1][0]   arr[1][1]   arr[1][2]
arr[2]  →   arr[2][0]   arr[2][1]   arr[2][2]
arr[3]  →   arr[3][0]   arr[3][1]   arr[3][2]
```

Shunday qilib, arr[4][3] ning e’lon qilinishi dasturda uchta turli xildagi obyektlarni yuzaga keltiradi: arr identifikatorli ko‘rsatkichni, to‘rtta ko‘rsatkichdan iborat nomsiz massivni va int turidagi o‘n ikkita sondan iborat nomsiz massivni. Nomsiz massivlarga kirish huquqiga ega bo‘lish uchun arr ko‘rsatkichli adresli ifodalar qo‘llanadi. Ko‘rsatkichlar massivi elementlariga kirish huquqi arr[2] yoki *(arr+2) shaklidagi indeksli ifodaning bittasini ko‘rsatish orqali amalga oshiriladi. Int turidagi ikki o‘lchamli sonlar massiviga kirish uchun arr[1][2] shaklidagi ikkita indeksli ifoda yoki unga ekvivalent bo‘lgan *(*(arr+1)+2) va (*(arr+1))[2] shaklidagi ifodalar qo‘llanishi kerak. Shuni ham hisobga olish kerakki, C tili sintaksisi nuqtayi nazaridan arr ko‘rsatkichi va arr[0], arr[1], arr[2], arr[3] ko‘rsatkichlari konstantalardir hamda ularning qiymatlarini dasturni

bajarish paytida o‘zgartirish mumkin emas. Uch o‘lchamli massivni joylashtirish ham xuddi shunga o‘xshash amalga oshiriladi hamda float arr3[3][4][5] ning e’lon qilinishi dasturda, float turidagi oltmishta sondan iborat uch o‘lchamli massivning o‘zidan tashqari, float turiga tuzilgan to‘rtta ko‘rsatkichdan iborat massivni, float ko‘rsatkichlar massiviga tuzilgan uchta ko‘rsatkichdan iborat massivni va float ga tuzilgan ko‘rsatkichlar massivining massivlariga ko‘rsatkichni yuzaga keltiradi. Ko‘p o‘lchamli massivlar elementlarini joylashtirishda ular xotirada satrlar bo‘yicha bir tartibda joylashtiriladi, ya’ni oxirgi indeks hammadan tezroq o‘zgaradi, birinchisi esa sekinroq o‘zgaradi. Bunday tartib ko‘p o‘lchamli massiv boshlang‘ich elementining adresini hamda faqat bitta indeks ifodasini qo‘llab, ko‘p o‘lchamli massivning har qanday elementiga murojaat qilish imkonini beradi.

Masalan, arr[1][2] elementiga murojaatni ptr2 ko‘rsatkichi yordamida amalga oshirsa bo‘ladi. Bu ko‘rsatkich esa ptr2[1*4+2] () murojaati yoki ptr2[6] murojaati sifatida int *ptr2=arr[0] shaklida e’lon qilingan bo‘ladi. Ta’kidlab o‘tish lozimki, tashqi tomondan o‘xshash arr[6] murojaatini bajarish mumkin emas, chunki 6 indeksli ko‘rsatkich mavjud emas.

Shuningdek, uch o‘lchamli massivga kiradigan arr3[2][3][4] elementiga murojaat uchun float *ptr3=arr3[0][0] ko‘rinishida tavsiflangan, ptr3[3*2+4*3+4] yoki ptr3[22] shaklidagi bitta indeksli ifodaga ega bo‘lgan ko‘rsatkichni qo‘llash mumkin.

Ko‘rsatkichlar massivlari

Ko‘rsatkichlar massivlari quyidagicha ta’riflanadi:

<tip> *<nom>[<son>]

Misol uchun int *pt[6] ta’rif int tipidagi obyektlarga olti elementli massivni kiritadi. Ko‘rsatkichlar massivlari satrlar massivlarini tasvirlash uchun qulaydir. Misol uchun familiyalar ro‘yxatini kiritish uchun ikki o‘lchovli massivdan foydalanish kerak: char fam[][20]={ ″Olimov″, ″Rahimov″, ″Ergashev″}

Bunday po‘yxat xotirada 60 elementdan iborat bo‘ladi, chunki har bir familiyagacha 0 lar bilan to‘ldiriladi. Ko‘rsatkichlar massivi yordamida bu massivni quyidagicha ta’riflash mumkin:

char *pf[]= {″Olimov″, ″Rahimov″, ″Ergashev″}.

Bu holda ro‘yxat xotirada 23 elementdan iborat bo‘ladi, chunki har bir familiya oxiriga 0 belgisi qo‘yiladi. Ko‘rsatkichlar massivlari murakkab elementlarni sodda usulda tartiblashga imkon beradi.

Ko‘rsatkichlar massivlari funksiyalarda matritsalar qiymatlarini o‘zgartirish uchun ishlatilishi mumkin. Quyidagi misolda matritsani transponirlash funksiyasi ishlatiladi:

```
#include <iostream.h>
void trans(int n,double *p[])
{ double x;
for (int i=0;i<n—1;i++)
for (int j=i+1;j<n;j++)
{x=p[i][j];
p[i][j]=p[j][i];
p[j][i]=x;
}
};
void main()
{double a[][3]={{11,12,13},{21,22,23},{31,32,33}};
double* ptr[3]={(double*)&a[0],(double*)&a[1], (double*)&a[2]};
int n=3;
trans(n,ptr);
for (int i=0;i<n;i++)
{cout<<″\n″ <<i+1;
for (int j=0;j<n;j++)
cout<<″  ″<<a[i][j];
};
};
```

Dinamik massivlar

C++ tilida o‘zgaruvchilar yo statik tarzda — kompilyatsiya paytida, yoki standart kutubxonadan funksiyalarni chaqirib olish yo‘li bilan dinamik tarzda — dasturni bajarish paytida joylashtirilishi mumkin. Asosiy farq ushbu usullarni qo‘llashda — ularning samaradorligi va moslashuvchanligida ko‘rinadi. Statik joylashtirish samaraliroq, chunki bunda xotirani ajratish dastur bajarilishidan oldin sodir bo‘ladi. Biroq bu usulning moslashuvchanligi ancha past, chunki bunda biz joylashtirilayotgan obyektning turi va o‘lchamlarini avvaldan bilishimiz kerak bo‘ladi. Masalan, matniy faylning ichidagisini satrlarning statik massivida joylashtirish qiyin: avvaldan uning o‘lchamlarini bilish kerak

bo‘ladi. Noma’lum sonli elementlarni oldindan saqlash va ishlov berish kerak bo‘lgan masalalar odatda xotiraning dinamik ajratilishini talab qiladi.

Xotirani dinamik va statik ajratish o‘rtasidagi asosiy farqlar quyidagicha:

■ statik obyektlar nomlangan o‘zgaruvchilar bilan belgilanadi hamda ushbu obyektlar o‘rtasidagi amallar to‘g‘ridan-to‘g‘ri, ularning nomlaridan foydalangan holda, amalga oshiriladi. Dinamik obyektlar o‘z shaxsiy otlariga ega bo‘lmaydi va ular ustidagi amallar bilvosita, ko‘rsatkichlar yordamida, amalga oshiriladi;

■ statik obyektlar uchun xotirani ajratish va bo‘shatish kompilyator tomonidan avtomatik tarzda amalga oshiriladi. Dasturchi bu haqda o‘zi qayg‘urishi kerak emas. Statik obyektlar uchun xotirani ajratish va bo‘shatish to‘laligicha dasturchi zimmasiga yuklatiladi. Bu anchayin qiyin masala va uni yechishda xatoga yo‘l qo‘yish oson.

Dinamik tarzda ajratilayotgan xotira ustida turli xatti-harakatlarni amalga oshirish uchun new va delete operatorlari xizmat qiladi.

Shu paytga qadar barcha misollarda statik xotira ajratish qo‘llanadi. Masalan, i o‘zgaruvchisini aniqlash:

```
int i=1024;
```

Bu buyruq xotirada shunday sohani ajratib beradiki, u int turidagi o‘zgaruvchini saqlash, ushbu soha bilan i nomini bog‘lash hamda u yerga 1024 qiymatini joylashtirish uchun yetarli bo‘ladi. Bularning hammasi dastur bajarilishidan oldin kompilyatsiya bosqichida amalga oshiriladi.

Biroq o‘zgaruvchiga xotirani ajratib berish uchun yana bir usul mavjud bo‘lib, u new operatorini qo‘llashdan iborat.

New operatori ikkita shaklga ega. Birinchi shakl ma’lum bir turdagi yakka obyektga xotirani ajratib beradi:

```
int*pint=new int(1024);
```

Bu yerda new operatori int turidagi nomsiz obyektga xotirani ajratib beradi, uni 1024 qiymati bilan nomlantiradi (initsiallashtiradi) hamda yaratilgan obyekt adresini qaytarib beradi. Bu adres pint ko‘rsatkichiga joylashtiriladi. Ushbu nomsiz obyekt ustidagi barcha xatti-harakatlar shu ko‘rsatkich bilan ishlash orqali amalga oshiriladi, chunki dinamik obyekt bilan to‘g‘ridan-to‘g‘ri ish olib borish (manipulyatsiyalar o‘tkazish) mumkin emas.

New operatorining ikkinchi shakli ma'lum bir turdagi elementlardan tashkil topgan berilgan o'lchamlardagi massivga xotira ajratib beradi:

```
int *pia=new int[4];
```

Bu misolda xotira int turidagi to'rtta elementdan iborat massivga xotira ajratiladi. Afsuski, new operatorining bu shakli massiv elementlarini nomlantirish (initsiallashtirish) imkonini bermaydi.

New operatorining har ikkala shakli ham bir xil ko'rsatkichni qaytarishi (keltirilgan misolda bu butun sonning ko'rsatkichi) ayrim chalkashliklarga olib keladi. pint ham, pia ham aynan bir xil e'lon qilingan. Biroq pint operatori int turidagi bitta obyektni ko'rsatadi, pia esa int turidagi to'rtta obyektdan iborat massivning birinchi elementini ko'rsatadi.

Dinamik obyekt kerak bo'lmay qolganda, unga ajratilgan xotirani to'g'ridan-to'g'ri bo'shatish kerak. Bu ish delete operatori yordamida amalga oshiriladi:

```
delete pint;
```

Obyektning bo'shatilishi, new kabi, ikkita shaklga ega — yakka obyekt uchun va massiv uchun:

```
delete[] pia;
```

Agar ajratilgan xotirani bo'shatish esdan chiqqudek bo'lsa, bu xotira bekordan-bekorga sarflana boshlaydi va foydalanilmay qoladi, biroq, agar uning ko'rsatkichi o'z qiymatini o'zgartirgan bo'lsa, uni tizimga qaytarish mumkin emas. Bu hodisa *xotiraning yo'qotilishi (утечка памяти)* degan maxsus nom bilan ataladi. Pirovard natijada dastur xotira yetishmagani tufayli avariya holatida tugallanadi (agar u ancha vaqt ishlayversa).

Belgili axborot va satrlar

C++ da belgili ma'lumotlar uchun char turi qabul qilingan. Belgili axborotni taqdim etishda belgilar, simvolli o'zgaruvchilar va matniy konstantalar qabul qilingan.

Misollar:

```
const char c='c';//belgi — bir baytni egallaydi, uning qiymati o'zgarmaydi.
```

char a,b;//belgili o‘zgaruvchilar, bir baytdan joy egallaydi, qiymatlari o‘zgaradi.

const char *s=″\n satrining misoli″;//matniy konstanta.

C++ dagi satr — bu nol — belgi - ′\0′ (nol-terminator) bilan tugallanuvchi belgilar massivi. Nol-terminatorning holatiga qarab satrning amaldagi uzunligi aniqlanadi. Bunday massivdagi elementlar soni, satr tasviriga qaraganda, bittaga ko‘p.

Qiymat berish operatori yordamida satrga qiymat berish mumkin emas. Satrni massivga yoki kiritish paytida, yoki nomlantirish yordamida joylashtirish mumkin.

Misol:

```
void main()
{
        char s1[10]=″string1″;
        int k=sizeof (s1);
       cout<<s1<<′\t′<<k<<endl;
        char s2[]=″string2″;
        k=sizeof(s2);
       cout<<s2<<′\t′<<k<<endl;
         char s3[]={′s′,′t′,′r′,′i′,′n′,′g′,′3′};
        k=sizeof(s3);
       cout<<s3<<′\t′<<k<<endl;
char *s4=″string4″;//satr ko‘rsatkichi, uni o‘zgartirib bo‘lmaydi.
        k=sizeof(s4);
       cout<<s4<<′\t′<<k<<endl;
}
```

Natijalar:

string1 10 — 10 bayt ajratilgan, shu jumladan \0 ga
string2 8 — 8 bayt ajratilgan (7+1 bayt \0 ga)
string3 8 — 8 bayt ajratilgan (7+1 bayt \0 ga)
string4 4 — ko‘rsatkichning o‘lchamlari

C tilida satrlar bilan ishlash funksiyalari

C tilida satrlar bilan ishlash uchun maxsus funksiyalar mavjud bo‘lib **string.h** faylida joylashgandir. Ba’zilarini ko‘rib chiqamiz:

unsigned **strlen**(const char*s)— satr uzunligini hisoblash;

int **strcmp**(const char*s1, const char *s2) — satrlarni solishtirish. Agar s1<s2 bo‘lsa natija<0; agar s1==s2 bo‘lsa natija=0; agar s2>s1 bo‘lsa natija>0.

int **strcnmp**(const char*s1, const char *s2) — satrlar birinchi n simvolini solishtirish.

char***strcpy**(char*s1, const char*s2) — s1 satrdan s2 satrga nusxa olish.

char***strncpy**(char*s1, const char*s2, int n) — s1 satr birinchi n simvolidan s2 satrga nusxa olish.

char***strcat**(char*s1, const char*s2) — s1 satrga s2 satrni ulash.

char***strncat**(char*s1, const char*s2) — s1 satrga s2 satrning birinchi n simvolini ulash.

2.6. Funksiyalar, massivlar, ko‘rsatkichlar

Bir o‘lchamli massivlarni funksiya parametrlari sifatida uzatish

Massivdan funksiya parametri sifatida foydalanganda, funksiyaning birinchi elementiga ko‘rsatkich uzatiladi, ya’ni massiv hammavaqt adres bo‘yicha uzatiladi. Bunda massivdagi elementlarning miqdori haqidagi axborot yo‘qotiladi, shuning uchun massivning o‘lchamlari haqidagi ma’lumotni alohida parametr sifatida uzatish kerak. Funksiyaga massiv boshlanishi uchun ko‘rsatkich uzatilgani tufayli (adres bo‘yicha uzatish), funksiya tanasining operatorlari hisobiga massiv o‘zgarishi mumkin.

Misol:

Massivdan barcha juft elementlar chiqarilsin:

```
#include <iostream.h>
#include <stdlib.h>
int form(int a[100])
{
int n;
    cout<<"\nEnter n";
    cin>>n;
    for(int i=0;i<n;i++)
          a[i]=rand()%100;
    return n;
}
void print(int a[100],int n)
{
    for(int i=0;i<n;i++)
          cout<<a[i]<<" ";
    cout<<"\n";
}
```

```
void Dell(int a[100],int&n)
{
    int j=0,i,b[100];
    for(i=0;i<n;i++)
            if(a[i]%2!=0)
            {
                    b[j]=a[i];j++;
            }
            n=j;
            for(i=0;i<n;i++)a[i]=b[i];
}
void main()
{
    int a[100];
    int n;
    n=form(a);
    print(a,n);
    Dell(a,n);
    print(a,n);
}
```

Satrlarni funksiyalar parametrlari sifatida uzatish

Satrlar funksiyaga char turidagi bir o‘lchamli massivlar sifatida yoki char* turidagi ko‘rsatkichlar sifatida uzatilishi mumkin. Oddiy massivlardan farqli o‘laroq, funksiyada satr uzunligi ko‘rsatilmaydi, chunki satr oxirida satr oxiri /0 belgisi bor.

Misol:

Berilgan belgini satrda qidirish funksiyasi

```
int  find(char *s,char c)
{
for (int i=0;i<strlen(s);i++)
if(s[I]==c) return i;
return —1
}
```

Funksiyaga ko‘p o‘lchamli massivlarni uzatish

Ko‘p o‘lchamli massivlarni funksiyaga uzatishda barcha o‘lchamlar parametrlar sifatida uzatilishi kerak. C va C++ da ko‘p o‘lchamli massivlar aniqlanishi bo‘yicha mavjud emas. Agar biz bir nechta indeksga ega bo‘lgan massivni tavsiflasak (masalan, int mas[3][4]),

bu degani, biz bir o‘lchamli mas massivini tavsifladik, bir o‘lchamli int [4] massivlarining ko‘rsatkichlari esa uning elementlaridir.

Misol:

Kvadrat matritsani uzatish (transpozitsiya qilish).

Agar void transp(int a[][],int n){.....} funksiyasining sarlavhasini aniqlasak, bu holda biz funksiyaga noma’lum o‘lchamdagi massivni uzatishni xohlagan bo‘lib qolamiz. Aniqlanishiga ko‘ra massiv bir o‘lchamli hamda uning elementlari bir xil uzunlikda bo‘lishi kerak. Massivni uzatishda uning elementlarining o‘lchamlari haqida ham biron narsa deyilmagan, shuning uchun kompilyator xato chiqarib beradi.

Bu muammoning eng sodda yechimi funksiyani quyidagicha aniqlashdir:

void transp(int a[][4],int n), bu holda har bir satr o‘lchami 4 bo‘ladi, massiv ko‘rsatkichlarining o‘lchami esa hisoblab chiqariladi.

```
#include<iostream.h>
const int N=4;//
void transp(int a[][N],int n)
{
int r;
for(int i=0;i<n;i++)
for(int j=0;j<n;j++)
if(i<j)
{
r[a[i][j];a[i][j]=a[j][i];a[j][i]=r;
}
}
void main()
{
int mas[n][n];
for(int i=0;i<N;i++)
for(int j=0;j<N; lj++)
cin>>mas[i][j];
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
cout<<mas[i][j]
cout<<″\n″;
}
transp(n,mas);
```

```
for(i=0;i<N;i++)
{
for(j=0;j<n;j++)
cout<<mas[i][j]
cout<<"\n";
}
}
```

Funksiya ko‘rsatkichi

Har bir funksiya qaytarilayotgan qiymat turi, nomi va funksiya parametrlari turlarining ro‘yxati bilan tavsiflanadi. Agar funksiya nomidan keyinchalik qavslarsiz va parametrlarsiz foydalanilsa, bu holda u ushbu funksiya ko‘rsatkichi sifatida amal qila boshlaydi, xotirada funksiyani joylashtirish adresi esa uning qiymati bo‘lib qoladi. Bu qiymatni boshqa ko‘rsatkichga ham berish mumkin bo‘ladi. Bu holda ushbu yangi ko‘rsatkichdan funksiyani chaqirib olish uchun foydalanish mumkin bo‘ladi. Funksiya ko‘rsatkichi quyidagicha aniqlanadi:

```
funksiya_turi(ko‘rsatkich_*nomi)(parametrlar spetsifikatsiyasi)
Misol:
int f1(char c){.....}//funksiyani aniqlash
int(*ptrf1)(char);//f1 funksiyasiga ko‘rsatkichni aniqlash
```

Ko‘rsatkichni aniqlashda parametrlarning miqdori va turi ko‘rsatkich o‘rnatilayotgan funksiyani aniqlashdagi tegishli turlarga mos kelishi kerak.

Ko‘rsatkich yordamida funksiyani chaqirish quyidagi ko‘rinishga ega:

```
(ko‘rsatkich_*nomi)(faktik parametrlar ro‘yxati)
```

Misol:

```
#include <iostream.h>
void f1()
(cout<<"\n funksiya f1";)
void f2()
{cout<<"\n funksiya f2";}
void main()
{
void(*ptr)();//funksiya ko‘rsatkichi
ptr=f2;//ko‘rsatkichga f2 funksiyasining adresi beriladi
(*ptr)();// f2 funksiyasini chaqirish
```

```
ptr=f1;//ko‘rsatkichga f1 funksiyasining adresi beriladi
(*ptr)();//ko‘rsatkich yordamida f1 funksiyasini chaqirish
}
```

Aniqlashda funksiya ko‘rsatkichi shu paytning o‘zidayoq nomlantirilishi mumkin:

```
void (*ptr)()=f1;
```

Funksiyalarga iqtiboslar

Funksiyaga ko‘rsatkich qanday aniqlansa, funksiyaga iqtibos ham xuddi shunday aniqlanadi:

```
funksiya_turi(&iqtibos_nomi)(parametrlar)nomlantiruvchi_ifoda;
```

Misol:

```
int(&fret)(float,int)=f;// iqtibosni aniqlash
```

Funksiya nomini parametrlarsiz va qavslarsiz qo‘llash funksiya adresi sifatida qabul qilinadi. Funksiyaga iqtibos funksiya nomining sinonimi bo‘ladi. Funksiyaga iqtibosning qiymatini o‘zgartirib bo‘lmaydi, shuning uchun ko‘p o‘rinda iqtibosga ko‘rsatkichlar emas, funksiyaga ko‘rsatkichlar qo‘llanadi.

Misol:

```
#include <iostream.h>
void f(char c)
{cout<<″\n″<<c;}
void main()
{
void(*pf)(char);//funksiya ko‘rsatkichi
void(&pf)(char);//iqtibos ko‘rsatkichi
f(′A′);//nomi bo‘yicha chaqirish
pt=f;//ko‘rsatkich funksiyaga qo‘yiladi
(*pt)(′B′);//ko‘rsatkich yordamida chaqirish
rf(′S′);//iqtibos bo‘yicha chaqirish
}
```

2.7. Tuzilmalar bilan ishlash

O‘zgaruvchilarning qo‘shimcha turlari, struct

Tuzilma — bu tarkibli obyekt bo‘lib, unga har qanday turdagi elementlar kiradi. Bir turdagi obyekt bo‘lgan massivdan farqli o‘laroq,

tuzilma bir turda bo‘lmasligi mumkin. Tuzilmaning turi quyidagi ko‘rinishdagi yozuv bilan aniqlanadi:

struct {aniqlovchilar ro‘yxati}

Tuzilmada aqalli bitta komponent albatta ko‘rsatilishi lozim. Tuzilmalarning aniqlovchisi quyidagi ko‘rinishga ega:

Ma’lumotlar_turi tavsiflagich;

Bu yerda ma’lumotlar_turi tavsiflagichlarda aniqlanadigan obyektlar uchun tuzilma turini ko‘rsatadi. Eng oddiy ko‘rinishda tavsiflagichlar identifikatorlar (nomlagichlar) yoki massivlarning o‘zginasidir.

Misol:

```
struct      str
{
        double x,y;
} s1, s2, sm[9];

struct
{
        int year;
        char, month, day;
} date1, date2;
```

s1 va s2 o‘zgaruvchilari tuzilmalar kabi aniqlanib, bu tuzilmalarning har biri ikkita x va y komponentlar (tarkibiy qismlar) dan iborat. Sm o‘zgaruvchisi to‘qqizta tuzilmadan iborat massiv sifatida aniqlanadi. Ikkita o‘zgaruvchi — date1 va date2 ning har bittasi uchta komponent — year, month, day dan iborat.

Nomni tuzilma turi bilan assotsiatsiyalashning boshqa usuli ham mavjud bo‘lib, u tuzilma turining nomidan foydalanishga asoslangan. Tuzilma turining nomi (teg) standart tur nomi bilan bir xil. Tuzilma tegi quyidagicha aniqlanadi:

struct teg {tavsiflar ro‘yxati;};

Bu yerda teg identifikator bo‘lib xizmat qiladi.

Keyingi misolda ham avvalgi misoldagi ma’lumotlarning o‘zginasi initsiatsiya (tashabbuslash) qilinadi, faqat endi tuzilma tegi qo‘llanadi.

Misol:

```
struct st
{
        double x, y;
};
st          s1, s2, sm[9];
struct Date
{
    int year;
    char, month, day;
}
Date        date1, date2;
```

Keyin student identifikatori tuzilmaning tegi sifatida tavsiflanadi:

```
struct student
{
AnsiString name;  //O‘quvchining familiyasi, ismi, otasining ismi
AnsiString group;       //Guruh
int phone;              //Uy telefoni
};
```

Tuzilma tegi ushbu turdagi tuzilmalarni keyinchalik e’lon qilishda quyidagi shaklda qo‘llanadi:

teg — identifikatorlarning ro‘yxati;

Misol:

```
student st1, st2;
```

Tuzilma teglaridan foydalanish rekursiv tuzilmalarning tavsifi uchun kerak. Quyida tuzilmalar rekursiv teglaridan foydalanish usuli ko‘rib chiqiladi:

```
struct node
{
        int data
        struct node * next;
} st1_node;
```

node tuzilmasining tegi haqiqatan ham rekursivdir, chunki u o‘zining shaxsiy tavsifida, ya’ni next ko‘rsatkichini rasmiylashtirishda (formalizatsiyalashda) qo‘llanadi. Tuzilmalar to‘g‘ridan-to‘g‘ri rekursiv bo‘lolmaydi, ya’ni node tuzilmasi node tuzilmasi bo‘lgan komponentaga ega bo‘lolmaydi. Biroq har qanday tuzilma o‘z turiga

ko‘rsatkich bo‘lgan komponentaga ega bo‘lishi mumkin. Keltirilgan misolda aynan shunday qilingan.

Tuzilma komponentalariga kirish tuzilma nomini hamda nuqtadan keyin kelgan ajratib ko‘rsatilgan komponenta nomini ko‘rsatish yordamida amalga oshiriladi. Masalan:

```
st1.name=″Tursunov″;
st1_node.data=st1.phone;
```

3. C++DA OBYEKTGA MO‘LJALLANGAN DASTURLASH

3.1. Obyektga mo‘ljallangan yondashuv (OMY) va C++ ning tamoyillari

Obyekt — mavhum (abstrakt) mohiyat bo‘lib, u bizni o‘rab turgan haqiqiy olamning tavsiflariga ega. Obyektlarni yaratish va ular ustida manipulyatsiyalar olib borish C++ tilining qandaydir alohida imtiyozi emas, balki obyektlarning tavsifi va ular ustida o‘tkaziladigan operatsiyalarni kodli konstruksiyalarda o‘zida mujassamlantiradigan dasturlash metodologiyasi (uslubiyoti) ning natijasidir. Dasturning har bir obyekti, har qanday haqiqiy obyekt kabi, o‘z atributlari va o‘ziga xos xulq-atvori bilan ajralib turadi. Obyektlarni turli kategoriyalarga ko‘ra tasniflash mumkin: masalan, mening «Cassio» raqamli qo‘l soatim soatlar sinfiga mansub. Soatlarning dasturiy realizatsiyasi (ishlatilishi), standart holat sifatida, sizning kompyuteringizning operatsiya tizimi tarkibiga kiradi.

Har bir sinf sinflar tabaqalanishida (ierarxiyasida) ma’lum o‘rinni egallaydi. Masalan, barcha soatlar vaqtni o‘lchash asboblari sinfiga (tabaqalanishda ancha yuqori turgan) mansub, soatlar sinfining o‘zi esa xuddi shu mavzudagi ko‘plab hosila variatsiyalarni o‘z ichiga oladi. Shunday qilib, har qanday sinf obyektlarning biror bir kategoriyasini aniqlaydi, har qanday obyekt esa biror bir sinf ekzemplyari (nusxasi)dir.

Obyektga mo‘ljallangan dasturlash (OMD) — bu dasturchining asosiy diqqatini obyektlarni ishlatish tafsilotlariga emas, balki obyektlar o‘rtasidagi aloqalarga qaratadigan metodika. Bu bobda OMD ning asosiy tamoyillari (inkapsulatsiyalash, vorislik, polimorfizm, sinflar va obyektlarning yaratilishi) C++ Builder tilining vizual ishlov berish integratsiyalangan muhitida qabul qilingan yangi tushuncha va atamalar bilan izohlanadi va to‘ldiriladi. Shuningdek, yangi imkoniyatlar (komponentalar, xususiyatlar, voqea-hodisaning qayta

ishlagichi) evaziga tilni kengaytirish tavsiflari hamda ANSI C++ standartining oxirgi qo‘shimchalari (shablonlar, nomlar fazosi, to‘g‘ridan-to‘g‘ri va noaniq e’lonlar, dasturni bajarishda turlar identifikatsiyasi, istisnolar)ning tavsiflari keltiriladi.

Bob umumiy xarakterga ega bo‘lib, OMD ning o‘quvchi keyinchalik duch kelishi mumkin bo‘lgan maxsus atamalari bilan tanishtirish maqsadini ko‘zlaydi. Buning zarurligi shundaki, C++ Builder tili OMD ning tipik (namunaviy) tizimi hisoblanadi hamda OMD ning rivojida yetakchi o‘rin egallashga da’vogarlik qiladi.

Inkapsulatsiyalash

Inkapsulatsiyalash — ma’lumotlarning va shu ma’lumotlar ustida ish olib boradigan kodlarning bitta obyektda birlashtirilishi. OMD atamachiligida ma’lumotlar obyekt *ma’lumotlari a’zolari (data members)* deb, kodlar *obyektli metodlar yoki funksiya—a’zolar (methods, member functions)* deb ataladi.

Inkapsulatsiyalash obyektni tashqi muhitdan maksimal darajada ajratish (izolyatsiya qilish) imkonini beradi. Bu ishlab chiqilayotgan dasturlarning ishonchliligini sezilarli darajada oshiradi, chunki obyektda mujassamlangan (lokallashtirilgan) funksiyalar dastur bilan nisbatan kam hajmdagi ma’lumotlarni almashinadi, buning ustiga ushbu ma’lumotlarning miqdori va turi odatda sinchkovlik bilan nazorat qilinadi. Natijada obyektda inkapsulatsiyalangan funksiyalar va ma’lumotlarning almashtirib qo‘yilishi yoki o‘zgartirilishi (modifikatsiya qilinishi) umuman olganda dastur uchun yaxshi kuzatib bo‘lmaydigan oqibatlarni olib kelmaydi (dasturlarning himoyalanganligini oshirish maqsadida OMD larda global o‘zgaruvchilar deyarli qo‘llanmaydi).

Inkapsulatsiyalashning yana bir muhim natijalaridan biri shundaki, bunda obyektlar almashinuvi, ularning bir dasturdan ikkinchisiga o‘tkazilishi osonlashadi. OMD da inkapsulatsiyalash tamoyilining soddaligi va qulayligi dasturchilarni C++ Builder tarkibiga kiruvchi Vizual Komponentlar Kutubxonasini kengaytirishga rag‘batlantiradi.

Sinflar, komponentlar va obyektlar

Sinf jismoniy mohiyatga ega emas, tuzilmaning e’lon qilinishi uning eng yaqin analogiyasidir. Sinf obyektni yaratish uchun qo‘llangandagina xotira ajralib chiqadi. Bu jarayon ham *sinf nusxasi (class instance)* ni yaratish deb ataladi.

C++ tilining har qanday obyekti bir xil atributlarga, shuningdek, ushbu sinfning boshqa obyektlari bilan funksionallikka ega. O‘z sinflarini yaratish hamda ushbu sinflar obyektlarining xulq-atvori uchun to‘liq mas’uliyat dasturchi zimmasiga yuklanadi. Biror bir muhitda ishlar ekan, dasturchi standart sinflarning kattagina kutubxonasi (masalan, C++ Builder Vizual Komponentalar Kutubxonasi) ga kirish huquqiga ega bo‘ladi.

Odatda, obyekt qandaydir unikal holatda mavjud bo‘lib, bu holat obyekt atributlarining joriy qiymatlari bilan belgilanadi. Obyekt sinfining funksionalligi ushbu sinf ekzemplyari (nusxasi) ustida o‘tkaziladigan operatsiyalar bilan belgilanadi.

C++ tilida sinfni aniqlash uchun ma’lumotlar a’zolari ustida ish olib boradigan hamda obyektlarning xulq-atvorini belgilaydigan ma’lumotlar a’zolari va metodlarini inkapsulatsiyalash talab qilinadi. Yuqoridagi misolimizga qaytib, shuni ta’kidlab o‘tamizki, «Casio» soatining suyuq kristali displeyi ushbu obyektning ma’lumotlar a’zosi bo‘ladi, boshqarish tugmachalari esa obyektli metodlar bo‘ladi. Soat tugmachalarini bosib, displeyda vaqtni o‘rnatish ishlarini olib borish mumkin, ya’ni OMD atamalarini qo‘llaydigan bo‘lsak, metodlar, ma’lumotlar a’zolarini o‘zgartirib, obyekt holatini modifikatsiya qiladi.

C++ Builder *komponentlar (components)* tushunchasini ham kiritadi. Komponentlar — maxsus sinflar bo‘lib, ularning xususiyatlari obyektlar atributlarini tashkil qiladi, ularning metodlari esa komponentli sinflarning tegishli nusxalari ustidagi operatsiyalarni amalga oshiradi. *Metod* tushunchasi odatda komponentli sinflar tarkibida qo‘llanadi va tashqi tomondan oddiy sinfning *funksiya-a’zo* atamasidan farq qilmaydi. C++Builder tili komponentlarning turi va funksional xulq-atvorini nafaqat metodlar yordamida, balki komponentlar sinflarigagina xos bo‘lgan xususiyatlar vositasida ham manipulyatsiya qilish imkonini beradi. C++ Builder muhitida ishlar ekansiz, siz ilovani loyihalash bosqichida ham, uni bajarish bosqichida ham komponentli obyekt ustida ishlash (manipulyatsiya qilish) mumkin ekanini tushunib yetishingiz ayon.

Komponentlar *xususiyatlari (propertion)* bu ma’lumotlar a’zolarining kengayishidir. Garchi ular ma’lumotlarni o‘z hollaricha saqlamasalar-da, biroq obyekt ma’lumotlari a’zolariga kirish huquqini ta’minlaydilar. Xususiyatlarni e’lon qilishda C++ Builder **_property** kalit-so‘zdan foydalanadi. *Voqealar (events)* yordamida komponenta o‘ziga qandaydir avvaldan belgilangan ta’sir ko‘rsatilganini foydalanuvchiga ma’lum qiladi. C++ Builder muhitida ishlab chiqilayotgan

dasturlardagi metodlar asosan ma'lum voqealarning yuzaga kelishida dastur reaksiyasini ularga nisbatan ishga soladigan *voqealarning qayta ishlagichlari (events handlers)* da qo'llanadi. Windows operatsiya tizimidagi voqealar va ma'lumotlardagi qandaydir o'xshashlikni payqab olish qiyin emas. Bu yerdagi oddiy tipik voqealar klaviaturadagi tugmacha yoki klavishalarni bosishdan iborat. Komponentalar o'z xususiyatlari, metodlari va voqealarini inkapsulatsiyalaydilar.

Bir qarashda, komponentalar C++ tilining boshqa obyektli sinflaridan, bir qator xususiyatlarni hisobga olmaganda, hech bir farq qilmaydi. Bu xususiyatlar orasida hozircha quyidagilarni ko'rsatib o'tamiz:

■ Komponentalarning ko'pchiligi interfeysning foydalanuvchi bilan boshqarish elementi bo'lib, ularning ayrimlari g'oyat murakkab xulq-atvorga ega.

■ Barcha komponentalar bitta umumiy ajdod — sinf (TComponent) ning bevosita yoki bilvosita avlodlaridirlar.

■ Komponentalar odatda bevosita qo'llanadi, ya'ni ularning xususiyatlari ustida ish olib boriladi (manipulyatsiyalar o'tkaziladi); ularning o'zlari yangi tarmoq sinflar (sinfchalar) qurish uchun bazaviy sinflar sifatida xizmat qila olmaydi.

■ Komponentalar faqat new operatori yordamida uyum (heap) ning dinamik xotirasida joylashtiriladi, oddiy sinflar obyektlarida bo'lganidek, stekda emas.

■ Komponentalar xususiyatlari RTTI — dinamik turlar identifikatsiyasini o'z ichiga oladi.

■ Komponentalarni Komponentalar Palitrasiga qo'shish va shundan so'ng C++Builder vizual ishlanmasining integrallashgan muhitiga tegishli Shakllar Muharriri vositasida ular ustida ishlash (manipulyatsiyalar o'tkazish) mumkin.

OMD obyektlarning o'zaro aloqasini talablarni biror bir obyektga yoki obyektlararo yuborish sifatida talqin etadi. Talabni olgan obyekt bunga tegishli usulni chaqirish bilan javob beradi. OMD ning SmallTalk kabi boshqa tillaridan farqli o'laroq, C++ tili «talab» tushunchasidan foydalanishni qo'llab-quvvatlamaydi. Talab — obyekt ustida qilinayotgan ish, metod esa kelib tushgan talabga obyektning javobi.

Yaqinroq olib qaralsa, metod — sinf ta'rifiga kiritilgan oddiy funksiya—a'zo. Metodni chaqirib olish uchun ushbu sinf kontekstida yoki biror bir voqeaning qayta ishlagichida funksiya nomini ko'rsatish kerak.

Aynan metodning sinf bilan yashirin aloqasi uni oddiy funksiya tushunchasidan ajratib turadi. Metodni bajarish paytida u o'z sinfining

barcha ma'lumotlariga kirish huquqiga ega bo'ladi, garchi ushbu sinf nomining ochiq-oydin spetsifikatsiyasini talab qilmasa ham. Bu ish beistisno har bir metodga yashirin parametrga, ya'ni sinf nusxasi (ekzemplyari) ga o'zgaruvchan ko'rsatkich this ni berish orqali ta'minlanadi. Har gal metod sinf ma'lumotlari a'zolariga murojaat qilganda, kompilyator this ko'rsatkichidan foydalanadigan maxsus kodni generatsiya qiladi.

Vorislik

Vorislik g'oyasi obyektlar xulq-atvorini modifikatsiyalash muammosini hal qiladi hamda OMD ga favqulodda kuch va moslashuvchanlik baxsh etadi. Vorislik, deyarli hech qanday cheklanishlarsiz, siz yoki boshqa biron kimsa tomonidan yaratilgan sinflarni izchil qurish va kengaytirish imkonini beradi. Eng oddiy sinflardan boshlab, murakkablik jihatidan asta-sekin ortib boradigan, ammo sozlanishi ham oson, ichki tuzilishi ham oddiy bo'lgan hosila sinflarni yaratish mumkin.

Ayniqsa yirik dasturiy loyihalarni ishlab chiqishda vorislik tamoyilini hayotga izchil tatbiq etish pasayib boruvchi tuzilmaviy dasturlash (umumiydan juz'iyga) texnikasi bilan yaxshi moslashadi hamda ko'p o'rinda bunday yondashuvni rag'batlantiradi. Bunda dastur kodining murakkabligi ancha kamayadi. Hosila sinf (avlod) o'z bazaviy sinfining (otasining) hamda sinflar tabaqalanishidagi o'zining barcha ajdodlarining hamma xususiyatlari, metodlari va voqealarini meros qilib oladi.

Vorislik paytida bazaviy sinf yangi atributlar va operatsiyalar hisobiga yanada o'sadi. Hosila sinfda odatda yangi ma'lumotlar a'zolari, xususiyatlar va metodlar paydo bo'ladi. Obyektlar bilan ishlashda dasturchi odatda konkret masalani hal qilish uchun eng to'g'ri keladigan sinfni tanlaydi hamda undan bitta yoki bir nechta voris avlod yaratadiki, ular o'z otalarida mavjud imkoniyatlardan ko'proq imkoniyatga ega bo'ladilar. *Do'stona* funksiyalar hosila sinfga barcha tashqi sinflar ma'lumotlari a'zolariga kirish huquqini olish imkonini beradilar.

Bundan tashqari, meros qilib olinayotgan metodlardan, ularning bazaviy sinfdagi ishi avlodga to'g'ri kelmasa, hosila sinf *ortiqcha yuklanishi (overload)* mumkin. OMD da ortiqcha yuklanishdan foydalanish har qanaqasiga rag'batlantiriladi, garchi bu so'zning to'g'ri ma'nosidan kelib chiqqanda, odatda ortiqcha yuklanishlar tavsiya qilinmaydi. Agar metod bittadan ortiq bir nomdagi funksiya bilan

assotsiatsiyalansa, u ortiqcha yuklangan deb aytiladi. E'tibor bering, sinflar tabaqalanishida ortiqcha yuklatilgan metodlarni chaqirib olish mexanizmi qayta aniqlangan funksiyalarni chaqirib olishdan mutlaqo farq qiladi. Ortiqcha yuklanish va qayta aniqlanish — bu turli tushunchalar. *Virtual metodlar* bazaviy sinf funksiyalarini qayta aniqlash uchun qo'llanadi.

Vorislik konsepsiyasini soat haqidagi misolga tatbiq qilish uchun, faraz qilaylik, vorislik tamoyiliga amal qilgan «Casio» firmasi soatning yangi modelini chiqarishga qaror qildi. Aytaylik, bu model tugmachalardan biri ikki marta bosilsa, vaqtni ovozda ayta oladi. Gapiradigan soatlar modeli (OMD atamalari bo'yicha, yangi sinf) ni yangidan yaratish o'rniga muhandislar ishni uning prototipidan boshlaydilar (OMD atamalari bo'yicha bazaviy sinfning yangi avlodini yaratadilar). Hosila obyekt otasining barcha atributlari va funksionalligini meros qilib oladi. Sintezlangan ovozda aytilgan sonlar avlodning yangi ma'lumotlar a'zolari bo'lib qoladi, tugmachalarning obyektli metodlari esa, ularning qo'shimcha funksionalligini ishga tushirish uchun ortiqcha yuklatilgan bo'lishi kerak. Tugmachalarning ikki marta bosilish hodisasiga yangi usul javob berib, u joriy vaqtga mos keladigan sonlar ketma-ketligi (yangi ma'lumotlar a'zolari) ning talaffuz qilinishida namoyon bo'ladi. Yuqorida aytilganlarning hammasi gapiradigan soatlarning dasturiy amalga oshirilishiga to'liq taalluqli.

3.2. Sinflarni ishlab chiqish

Sinflar ma'lum maqsadlarga erishish uchun ishlab chiqiladi. Odatda dasturchi mavhum g'oyadan boshlaydi va u asta-sekin loyihani ishlab chiqish jarayonida turli detallar bilan to'ldirib boriladi. Ba'zida bir-biriga g'oyat o'xshash bo'lgan bir necha sinfni ishlab chiqish bilan ish tugallanadi. Sinflarda kodlarni bu kabi takrorlash (dubllashtirish) dan qochish uchun bu sinflarni ikki qismga bo'lish kerak, ya'ni umumiy qismni ota sinfida aniqlab, farqlanadiganlarini hosila sinfda qoldirish kerak.

Sinfdan foydalanishdan oldin u e'lon qilinishi kerak. Odatda, amaliyotchi dasturchi tayyor bazaviy sinflardan foydalanadi, bundan tashqari u barcha spetsifikatsiyalarni va ichki ishlash yo'llarini bilishi mutlaqo shart emas. Biroq, C++bazaviy sinfidan foydalanishingiz uchun qanday ma'lumotlar a'zolari va metodlarga kira olishingiz mumkinligini (C++ Builder komponentasi qo'llansa, taqdim etilayotgan xususiyatlar va voqealarni ham) albatta bilishingiz lozim.

Bazaviy sinfni e'lon qilish

C++ Builder sizga o'z xususiyatlari, ma'lumotlari, metodlari va voqealari nomlarini inkapsulatsiyalaydigan bazaviy sinfni e'lon qilish imkonini beradi. O'zlarining bevosita vazifalarini bajarishdan tashqari, obyektli metodlar sinf xususiyatlari va ma'lumotlari qiymatlariga kirish uchun ma'lum imtiyozlarga ham ega bo'ladilar.

Sinf ichidagi har bir e'lon qilish, sinf nomi qaysi seksiyada paydo bo'lishiga qarab, bu nomlarga kirish imtiyozlarini aniqlaydi. Har bir seksiya quyidagi kalit-so'zlarning biridan boshlanadi: **private, protected, public.** Bazaviy sinfni e'lon qilishning umumlashma sintaksisi quyidagi ko'rinishga ega:

class className
private:
<privat ma'lumotlar a'zolari> <privat konstruktorlar> <privat metodlar>
protected:
<himoyalangan ma'lumotlar a'zolari> <himoyalangan konstruktorlar> <himoyalangan metodlar>
public:
<ommaviy xususiyatlar> <ommaviy ma'lumotlar a'zolari> <ommaviy konstruktorlar> <ommaviy destruktor> <ommaviy metodlar>

Shunday qilib, C++da bazaviy sinfni e'lon qilish quyidagicha kirish huquqlari va tegishli ko'rishlik sohasini taqdim etadi:

■ Privat **private** nomlar faqat ushbu sinf metodlariga ruxsat etilgan eng cheklangan kirish huquqiga ega. Hosila sinflar uchun bazaviy sinflarning privat metodlariga kirish taqiqlangan.

■ Himoyalangan **protected** nomlar ushbu sinf va undan hosil bo'lgan sinflar metodlariga ruxsat etilgan kirish huquqiga ega.

■ Ommaviy **public** nomlar barcha sinflar va ularning obyektlari metodlariga ruxsat etilgan cheksiz kirish huquqiga ega.

Quyidagi qoidalar sinf e'lon qilinishining turli seksiyalarining hosil bo'lishida qo'llaniladi:

1. Seksiyalar har qanday tartibda ham paydo bo'lishi, ularning nomlari esa takroran uchrayverishi mumkin.

2. Agar seksiya nomlanmagan bo'lsa, sinf nomlarining keyingi e'lonlarini kompilyator privat deb hisoblaydi. Bu yerda sinf va tuzilmaning e'lon qilinishida farq yuzaga kelyapti: tuzilma yashirin holda ommaviy deb olib qaraladi.

3. Agar siz haqiqatan ham ma'lumotlar a'zolariga har qayerdan kirishni ruxsat etmoqchi bo'lmasangiz, imkon darajasida ularni ommaviy seksiyaga joylashtirmang. Faqat hosila sinflar metodlariga kirish huquqini berish uchun ularni odatda himoyalangan deb e'lon qiladilar.

4. Metodlardan ma'lumotlar xususiyatlari va a'zolarini tanlash, tekshirish va qiymatlarini o'rnatish uchun foydalaning.

5. Konstruktorlar va destruktorlar maxsus funksiyalar bo'lib, qiymatni qaytarmaydilar va o'z sinfining nomiga ega bo'ladilar. Konstruktor berilgan sinf obyektini quradi, destruktor esa uni olib tashlaydi.

6. C++ning bittadan ortiq yo'riqnomasiga ega bo'lgan metodlarni (konstruktorlar va destruktorlar kabi) sinfdan tashqarida deb e'lon qilish tavsiya etiladi.

Navbatdagi misolda bazaviy sinfning e'lon qilinishini biror bir aniq taxmin bilan to'ldirishga harakat qilingani ko'rsatilgan. Shuni ta'kidlab o'tish lozimki, C++ Builder sinfi komponentlari uchun Count xususiyatini himoyalangan seksiyada e'lon qilish xos bo'lsa, FCount ma'lumotlar a'zosiga yozuvni amalga oshiradigan SetCount metodini privat seksiyada e'lon qilish xos.

```
Class Tpoint {private:
int FCount; //Ma'lumotlarning privat a'zosi
void _fastcall SetCount(int Value);
protected:
_property int Count=//Himoyalangan xususiyat
{read=Fcount, write=SetCount}:
double x;//Himoyalangan ma'lumotlar a'zosi
double y;//Himoyalangan ma'lumotlar a'zosi
public:
Tpoint(double xVal, double yVal)://Konstruktor
double getX();
double getY();
```

Metodlarning e'lon qilinishi va aniqlanishi turli fayllarda saqlanadi. Misollar shuni ko'rsatadiki, metodlar sinfdan tashqarida aniqlanganda ularning nomlarini kvalifikatsiya qilish (ixtisoslashtirish) kerak. Metodning ko'rimlilik sohasini aniqlaydigan uning bunday kvalifikatsiya sintaksisi quyidagi ko'rinishga ega:

<Sinf nomi>::<metod nomi>

Siz sinfni e'lon qilganingizdan keyin uning nomidan ushbu sinf obyektini e'lon qilishda identifikator turi sifatida foydalanish mumkin. Masalan:

TPoint* MyPoint.

Sinf metodlarini sinfdan tashqarida aniqlash

Funksiya prototipi ichida joylashtirilgan employee sinfini ko‘rib chiqamiz. Funksiyaning o‘zi esa sinfdan tashqarida aniqlangan.

Navbatdagi CLASSFUN.CPP dasturi show_employee funksiyasining ta’rifini sinfdan tashqarida joylashtiradi va bunda sinf nomini ko‘rsatish uchun global ruxsat operatoridan foydalanadi:

```
#include <iostream.h>
#include <string.h>
class employee
{
public:
  char name [64];
   long employee_id;
   float salary;
 void show_employee(void);
};
void employee::show_employee(void)
{
  cout << ″Ism: ″ << name << endl;
  cout << ″Tartib raqami:″ << employee_id << endl;
  cout << ″Maosh:″ << salary << endl;
};
void main(void)
{
  employee worker, boss;
 strcpy(worker.name, ″John Doe″);
   worker.employee_id = 12345;
 worker.salary = 25000;
 strcpy(boss.name, ″Happy Jamsa″);
boss.employee_id = 101;
 boss.salary = 1011012.00;
worker.show_employee();
boss.show_employee();
}
```

Konstruktorlar va destruktorlar

Nomlaridan ko‘rinib turganidek, konstruktor — bu metod bo‘lib, u o‘z xotirasida ushbu sinf obyektini quradi, destruktor esa — bu obyektni olib tashlaydigan metod. Konstruktorlar va destruktorlar boshqa obyektli metodlardan quyidagi xususiyatlariga ko‘ra farqlanadi:

■ O‘z sinfi nomi bilan bir xil bo‘lgan nomga ega.

■ Qaytariladigan qiymatga ega emas.

■ Garchi hosila sinf bazaviy sinflarning konstruktorlari va destruktorlarini chaqira olsa-da, konstruktor va destruktorlarning o‘zlari vorislik qilolmaydi.

■ Agar boshqacha e’lon qilinmagan bo‘lsa, kompilyator tomonidan avtomatik tarzda public sifatida generatsiya qilinadi.

■ Sinf obyektlarining yaratilishi va yo‘q qilinishini tegishli tarzda kafolatlash uchun kompilyator tomonidan chaqirib olinadi.

■ Agar obyekt dinamik xotiraning ajratilishi va yo‘q qilinishini talab qilsa, new va delete operatorlariga noaniq murojaatga ega bo‘lishi mumkin.

Quyida konstruktorlar va destruktorlar e’lonining umumlashma sintaksisini namoyish qiluvchi misol keltiramiz:

```
class className
{public:
//className ma’lumotlarining boshqa a’zolari; className (<parametrlar ro‘yxati;-)//Dalillarga ega konstruktor
className(const className&);//Nusxa ko‘chirish konstruktori
//Boshqa konstruktorlar
~className();//Destruktor
//Boshqa metodlar};
```

Sinf soni cheklanmagan konstruktorlarga ega bo‘lishi, shu jumladan, konstruktorlarga umuman ega bo‘lmasligi mumkin. Konstruktorlar virtual deb e’lon qilinishi mumkin emas. Hamma konstruktorlarni himoyalangan seksiyaga joylashtirmang hamda yashirin argumentlar qiymatidan foydalanib, ularning sonini kamaytirishga intiling.

Ko‘p hollarda sinf obyektlarini initsiallash (nomlash) ning bir nechta usullariga ega bo‘lish yaxshi natija beradi. Bunga bir nechta konstruktor vositasida erishish mumkin. Masalan:

```
class date {
    int month, day, year;
public:
    // ...
    date(int, int, int);   //yilning oyi, kuni
date(char*);              //satr ko‘rinishidagi sana
date(int);                 //bugungi kun, oy, yil
date();                    //yashirin sana: bugungisi
};
```

Ortiqcha yuklangan funksiyalar qanday qoidalarga amal qilsa, konstruktorlar ham parametrlar turlariga nisbatan xuddi shunday qoidalarga amal qiladilar. Agar konstruktorlar o‘z parametrlari turlari bo‘yicha ancha-muncha farq qilsa, kompilyator har gal foydalanganda to‘g‘ri parametrni tanlab olishi mumkin:

```
date today(4);
date july4(″1983-yil, 4-iyul″);
date guy(″5-noyabr″)
date now;          //yashirish initsiallashadi (nomlanadi)
```

Konstruktorlarning uch turi mavjud:

■ *Yashirin konstruktor* parametrlarga ega emas. Agar sinf bitta ham konstruktorga ega bo‘lmasa, kompilyator avtomatik tarzda bitta yashirin konstruktor yaratadiki, u o‘z sinfiga mansub obyektni yaratishda xotirani shunchaki ajratib beradi.

Date holatida har bir parametr uchun «yashirin qabul qilish: today» (bugun) sifatida talqin qilinadigan yashiringan qiymatni berish mumkin:

```
class date {
      int month, day, year;
  public:
      // ...
      date(int d =0, int m =0, int y =0);
      date(char*);          // satr vositasida berilgan sana
  };
date::date(int d, int m, int y)
  {
      day = d ? d : today.day;
      month = m ? m : today.month;
      year = y ? y : today.year;
      // yo‘l qo‘yiladigan sana ekanini tekshirish
      // ...
  }
```

■ *Dalillarga ega konstruktor* obyektning yaratilish paytida uni initsiallash (nomlash) ga, ya’ni turli funksiyalarni chaqirib olishga, dinamik xotirani ajratish, o‘zgaruvchilarga dastlabki qiymatlarni berish va h.k. ga imkon beradi.

■ *Nusxa ko‘chirish konstruktori* berilgan sinf obyektlarini yaratish uchun ma’lumotlarni ushbu sinfning mavjud bo‘lgan boshqa obyektidan nusxa ko‘chirish yo‘lidan borishga mo‘ljallangan. Bunday konstruktorlar ma’lumotlarning dinamik tuzilmalarini modellashtiradigan obyektlar nusxasini yaratishda ayniqsa maqsadga muvofiqdir. Biroq yashirin holda kompilyator yuzaki nusxalash konstruktorlari (shallow copy constructors) deb ataluvchi faqat ma’lumotlar a’zolaridan nusxa oladigan konstruktorlarni yaratadi. Shuning uchun, agar biror bir ma’lumotlar a’zolari ko‘rsatkichlarga ega bo‘lsa, ma’lumotlarning o‘zidan nusxa ko‘chirilmaydi. «Chuqur» nusxalashni konstruktor kodiga joriy qilish uchun tegishli yo‘riqnomalarni kiritish kerak.

Misol:

```
class string
        {
        char    *Str;
        int     size;
public:
        string(string&);        // nusxa ko‘chirish konstruktorlari
};
string::string(string& right) // dinamik o‘zgaruvchilar va resurslar
        {                       // nusxalarini yaratadi
        s = new char[right->size];
        strcpy(Str,right->Str);
        }
```

Sinf faqat bitta ommaviy destruktorni e’lon qilishi mumkin. Uning o‘z sinfi bilan bir xil bo‘lgan nomi oldidan ~ (tilda) belgisi turishi kerak. Destruktor parametrlarga ega emas hamda virtual deb e’lon qilinishi mumkin. Agar sinf destruktor e’loniga ega bo‘lmasa, kompilyator avtomatik ravishda uni yaratadi.

Odatda tegishli konstruktorlar bajargan operatsiyalarga teskari operatsiyalarni destruktorlar bajaradi. Agar siz fayl sinfi obyektini yaratgan bo‘lsangiz, bu holda destruktorda bu faylning yopilishi ehtimoldan xoli emas. Agar sinf konstruktori ma’lumotlar massivi uchun dinamik xotirani ajratsa (new operatori yordamida), bu holda destruktor ajratilgan xotirani bo‘shatishi (delete operatori yordamida) ehtimoldan xoli emas va h.k.

date obyekti konstruktori va destruktoriga misol:

```
class date { int *day, *month, *year
```

```
  public:
  date(int d, int m, int y)
  {
   day=new int;
   month=new int;
   year=new int;
   *day= d ? d : 1;
   *month = m ? m : 1;
   *year = y ? y : 1;
  }
...
~date()
  {
   delete day;
   delete month;
   delete year;
  } };
```

this ko‘rsatkichi

Obyekt uchun chaqirilgan funksiya (a’zo) da ma’lumotlar (obyekt a’zolari) ga bevosita iqtibos qilish mumkin. Masalan:

```
class x {
      int m;
  public:
      int readm() { return m; }
  };
x aa;
x bb;
  void f()
  {
      int a = aa.readm();
      int b = bb.readm();
      // ...
  }
```

readm() a’zosining birinchi chaqirilishida m aa.m ga tegishli bo‘ladi, ikkinchi chaqirilishida esa bb.m ga tegishli bo‘ladi.

Obyekt ko‘rsatkichi (a’zo funksiyasi mana shu obyekt uchun chaqirilgan) funksiyaning yashirin parametridir. Mana shu yashirin parametrga, xuddi this da bo‘lganidek, ochiq iqtibos qilish mumkin. X sinfining har bir funksiyasida this ko‘rsatkichi:

x* this;
sifatida yashirin tavsiflangan hamda shunday nomlangan (initsial-lashgan) ki, u obyektni ko‘rsatadi (funksiya—a’zo mana shu obyekt uchun chaqirilgan). X sinfini ekvivalent ravishda quyidagicha tavsiflash mumkin:

```
class x {
        int m;
public:
       int readm() {return this->m;}
};
```

A’zolarga iqtibos qilganda this dan foydalanish ortiqcha. This asosan funksiya-a’zolarni yozishda qo‘llanadi (bu funksiya-a’zolar bevosita ko‘rsatkichlar ustida ish olib boradi).

Konstantali funksiya-a’zolar va konstantali obyektlar

Biror bir funksiya yoki sinfga sinf obyektining shaxsiy qismiga kirish ruxsati talab qilinsa, ba’zida kirish huquqi qoidalaridan istisno tariqasida chetlanish talab qilinadi. Bu esa sinfning o‘zi o‘z obyektlariga «chetdan» kirish huquqlarini belgilaydi degan tamoyilga mos keladi. Kirishni nazorat qilish vositalariga funksiya-a’zolarning *konstantali (const)* deb e’lon qilinishi kiradi. Bu holda ular o‘zlari birga chaqirilayotgan joriy obyekt qiymatlarini o‘zgartirish huquqiga ega emaslar. Shunda bunday funksiyaning sarlavhasi quyidagi ko‘rinishga ega bo‘ladi:

```
void dat::put() const
    {...}
```

Konstanta obyektlarini ham xuddi shunday aniqlash mumkin:

```
const class a{...} value;
```

3.3. Statik elementlar va funksiyalar

Ma’lumotlar elementidan birgalikda foydalanish

Odatda, ma’lum sinf obyektlari yaratilayotganda, har bir obyekt o‘z-o‘zining ma’lumotlar elementlari to‘plamini oladi. Biroq shunday hollar ham yuzaga keladiki, unda bir xil sinflar obyektlariga bir yoki bir nechta ma’lumotlar elementlaridan (statik ma’lumotlar elementlaridan) birgalikda foydalanish kerak bo‘lib qoladi. Bunday

hollarda ma'lumotlar elementlari umumiy yoki juz'iy deb e'lon qilinadi, keyin esa tur oldidan, quyida ko'rsatilganidek, static kalit—so'z keladi:

```
private;
static int shared_value;
```

Sinf e'lon qilingach, elementni sinfdan tashqaridagi global o'zgaruvchi sifatida e'lon qilish kerak. Bu quyida shunday ko'rsatilgan:

```
int class_name::shared_value;
```

Navbatdagi SHARE_IT.CPP dasturi book_series sinfini aniqlaydi. Bu sinf (seriya)ning barcha obyektlari (kitoblari) uchun bir xilda bo'lgan page_count elementidan birgalikda foydalanadi. Agar dastur ushbu element qiymatini o'zgartirsa, bu o'zgarish shu ondayoq barcha sinf obyektlarida o'z aksini topadi:

```
#include <iostream.h>
#include <string.h>
class book_series
{
public:
 book_series(char *, char *, float);
 void show_book(void);
 void set_pages(int) ;
private:
 static int page_count;
  char title[64];
 char author[ 64 ];
 float  price;
};
int book_series::page__count;
void book_series::set_pages(int pages)
{
 page_count = pages;
}
book_series::book_series(char *title, char *author, float price)
{
 strcpy(book_series::title, title);
```

```
 strcpy(book_series::author, author);
book_series::price = price;
}
void book_series:: show_book (void)
{
 cout << ″Eslatma: ″ << title << endl;
 cout << ″Muallif: ″ << author << endl;
 cout << ″Narx: ″ << price << endl;
 cout << ″Betlar: ″ << page_count << endl;
}
void main(void)
{
 book_series programming( ″ C++ da dasturlashni o‘rganyapmiz″,
″Jamsa″, 213.95);
   book_series word( ″ Windows uchun Word bilan ishlashni
o‘rganyapmiz″, ″Wyatt″, 19.95);
word.set_pages(256);
 programming.show_book ();
 word.show_book() ;
  cout << endl << ″page_count ning o‘zgarishi ″ << endl;
programming.set_pages(512);
programming.show_book();
word.show_book();
}
```

Ko‘rinib turganidek, sinf *page_count* ni *static int* sifatida e’lon qiladi. Sinfni aniqlagandan so‘ng, dastur shu vaqtning o‘zida page_count elementini global o‘zgaruvchi sifatida e’lon qiladi. Dastur page_count elementini o‘zgartirganda, o‘zgarish shu vaqtning o‘zida-yoq book_series sinfining barcha obyektlarida namoyon bo‘ladi.

Agar obyektlar mavjud bo‘lmasa, public static atributli elementlardan foydalanish

Sinf elementini *static* kabi e’lon qilishda bu element ushbu sinfning barcha obyektlari tomonidan birgalikda qo‘llanadi. Biroq shunday vaziyatlar yuz berishi mumkinki, dastur hali obyektni yaratganicha yo‘q, ammo u elementdan foydalanishi kerak. Elementdan foydalanish uchun dastur uni public va static sifatida e’lon qilishi kerak. Masalan, USE_MBR.CPP dasturi, hatto agar book_series sinfidagi obyektlar mavjud bo‘lmasa ham bu sinfning page_count elementidan foydalanadi:

```
#include <iostream.h>
#include <string.h>
class book_series
{
public:
static int page_count;
private:
char title [64];
char author[64];
 float price;
};
int book_series::page_count;
void main(void)
{
 book_series::page_count = 256;
cout << "page_count ning joriy qiymati " << book_series::
page_count << endl;
}
```

Bu o‘rinda, sinf page_count sinfi elementini public sifatida e’lon qilgani uchun, hatto agar book_series sinfidagi obyektlar mavjud bo‘lmasa ham, dastur sinfning ushbu elementiga murojaat qilishi mumkin.

Static funksiya-elementlardan foydalanish

Avvalgi dastur ma’lumotlar *static* elementlarining qo‘llanishini ko‘rsatib bergan edi. C++xuddi shunday usul bilan *static* funksiya-elementlar (metodlar)ni aniqlash imkonini beradi. Agar *static* metod yaratilayotgan bo‘lsa, dastur bunday metodni, hatto uning obyektlari yaratilmagan holda ham, chaqirib olishi mumkin. Masalan, agar sinf sinfdan tashqari ma’lumotlar uchun qo‘llanishi mumkin bo‘lgan metodga ega bo‘lsa, siz bu usulni statik qila olishingiz mumkin bo‘lardi. Quyida displey ekranini tozalash uchun ANSI drayverining esc ketma-ketligidan foydalanadigan menu sinfi keltirilgan. Agar tizimda ANSI.SYS drayveri o‘rnatilgan bo‘lsa, ekranni tozalash uchun clear_screen metodidan foydalanish mumkin. Bu metod *static* deb e’lon qilingani tufayli, hatto agar menu turidagi obyekt mavjud bo‘lmasa ham, dastur undan foydalanishi mumkin. Keyingi CLR_SCR.CPP dasturi displey ekranini tozalash uchun clear_screen metodidan foydalanadi:

```
#include <iostream.h>
class menu
{
public:
   static void clear_screen(void);
   // Bu yerda boshqa metodlar bo‘lishi kerak
private:
 int number_of_menu_options;
};
void menu::clear_screen(void)
{
cout << '\033' << "[2]";
}
void main(void)
{
menu::clear_screen();
}
```

Dastur clear_screen elementini *static* sifatida e’lon qilar ekan, hatto agar menu turidagi obyekt mavjud bo‘lmasa ham dastur bu funksiyadan ekranni tozalash uchun foydalanishi mumkin. clear_screen funksiyasi ANSI Esc[2] esc ketma-ketligidan ekranni tozalash uchun foydalanishi mumkin.

3.4. Hosila sinflarni e’lon qilish

C++Builder o‘zining barcha ajdodlarining xususiyatlari, ma’lumotlari, metodlari va voqealarini meros qilib oladigan hosila sinfini e’lon qilish imkoniyatini beradi, shuningdek, yangi tavsiflarni e’lon qilishi hamda meros sifatida olinayotgan ayrim funksiyalarni ortiqcha yuklashi mumkin. Bazaviy sinfning ko‘rsatib o‘tilgan tavsiflarini meros qilib olib, yangi tug‘ilgan sinfni ushbu tavsiflarni kengaytirish, toraytirish, o‘zgartirish, yo‘q qilish yoki o‘zgarishsiz qoldirishga majburlash mumkin.

Vorislik bazaviy sinf kodidan hosila sinf nusxalarida takroran foydalanish imkonini beradi. *Takroran qo‘llash* konsepsiyasi jonli tabiatda o‘z paralleliga ega: DNK ni bazaviy material sifatida olib qarash mumkin, u har bir yaratilgan mavjudotdan o‘zining shaxsiy turini qayta ishlab chiqish uchun takroran foydalanadi.

Hosila sinfni e’lon qilishning umumlashgan sintaksisini ko‘rib chiqamiz. Seksiyalarni sanab o‘tish tartibi himoya imtiyozlarini eng

cheklanganlaridan to eng ommaviylariga qadar kengayib borishiga mos keladi:

class className: [<kirish huquqini beruvchi spetsifikator>] parent Class {
<Do‘stona sinflarni e’lon qilish>
private:
<privat ma’lumotlar a’zolari>
<privat konstruktorlar>
<privat metodlar> **protected:**
<himoyalangan ma’lumotlar a’zolari>
<himoyalangan konstruktorlar>
<himoyalangan metodlar> **public:**
<ommaviy xususiyatlar>
<ommaviy ma’lumotlar a’zolari>
<ommaviy konstruktorlar>
<ommaviy destruktor>
<ommaviy metodlar> **_published:**
<hammaga ma’lum bo‘lgan xususiyatlar>
<hammaga ma’lum bo‘lgan ma’lumotlar a’zolari>
<Do‘stona funksiyalarni e’lon qilish>

To‘ldiruvchi o‘rnidagi _published kalit-so‘zga ega bo‘lgan yangi seksiyaning paydo bo‘lganini ham ta’kidlab o‘tamiz. Bu so‘zni komponentli sinflarning keng ma’lum bo‘lgan elementlarini e’lon qilish uchun ANSI C++ standartiga C++Builder kiritadi. Bu seksiya ommaviy seksiyadan faqat shunisi bilan farqlanadiki, bunda kompilyator obyektning xususiyatlari, ma’lumotlar a’zolari va metodlari haqidagi RTTI axborotini generatsiya qiladi hamda C++Builder bu axborotning dastur bajarilayotgan paytda obyektlar Inspektoriga uzatilishini tashkil qiladi. O‘zining bevosita vazifasini bajarish imkoniga ega bo‘lishdan tashqari, obyektli metodlar yana boshqa sinflar xususiyatlarining qiymatlari va ma’lumotlariga ma’lum darajada kirish imtiyozlariga ham ega bo‘ladilar.

Sinf o‘zining bazaviy sinfidan yuzaga kelayotganida, uning barcha nomlari hosila sinfda avtomatik tarzda yashirin privat bo‘lib qoladi. Ammo uni, bazaviy sinfning quyidagi kirish spetsifikatorlarini ko‘rsatgan holda, osongina o‘zgartirish mumkin:

■ private. Bazaviy sinfning meros bo‘lib o‘tayotgan (ya’ni himoyalangan va ommaviy) nomlari hosila sinf nusxalarida kirib bo‘lmaydigan bo‘lib qoladi.

■ public. Bazaviy sinf va uning ajdodlarining nomlari hosila sinf nusxalarida kirib bo‘ladigan bo‘ladi, barcha himoyalangan nomlar esa himoyalangan bo‘lib qolaveradi.

Bazaviy sinf imkoniyatlarini *kengaytiradigan* sinflarni yuzaga keltirish mumkin: bu yo‘l siz uchun g‘oyat qulay, ammo ozgina ishlashni talab qilgan funksiyaga ega. Hosila sinfda kerakli funksiyani yangidan yaratish vaqtni bekorga sarflash bilan barobar. Buning o‘rniga bazaviy sinfda koddan takroran foydalanish kerak: bunda u talab qilingan darajada kengaytirilishi mumkin. Hosila sinfda sizni qoniqtirmaydigan bazaviy sinf funksiyasini qayta aniqlang, xolos.

Xuddi shunday yo‘l bilan bazaviy sinf imkoniyatlarini *cheklaydigan* sinflarni yuzaga keltirish mumkin. Bu yo‘l siz uchun g‘oyat qulay, ammo nimanidir ortiqcha qiladi.

Tugmacha obyekti turlaridan birining yaratilishi misolida tavsiflarni kengaytirish va cheklash metodikalarining qo‘llanishini ko‘rib chiqamiz: bu yerdagi obyekt turlaridan biri tipik hosila sinfi bo‘lib, u C++Builder ning Vizual Komponentalar Kutubxonasidagi TButtonControl bazaviy komponentasini meros qilib olish paytida yuzaga keladi. Har xil turdagi tugmachalar sizning dasturlaringiz grafik interfeysining dialogli darchalarida tez-tez paydo bo‘lib turadi.

TButtonControl bazaviy sinfi Draw ota usuli yordamida tugmachani biri ikkinchisining ichiga joylashtirilgan ikkita to‘g‘ri to‘rtburchak ko‘rinishida aks ettirish imkoniga ega: tashqi ramka va ichki bo‘yalgan soha.

Ramkasiz oddiy tugmachani yaratish uchun ota sinf sifatida TButtonControl dan foydalanib, SimpleButton hosila sinfini qurish hamda Draw metodini, uning funksionalligini cheklagan holda, qayta yuklash lozim:

```
class SimpleButton: public : TButtonControl { public:
SimpleButton(int x, int y) ;
void Draw() ;
~SimpleButton() { }
};
SimpleButton::SimpleButton(int x, int y) :
TButtonControl(x, y) { }
void SimpleButton::Draw()
{ outline->Draw();
}
```

SimpleButton uchun konstruktor obyektining yagona vazifasi — ikki parametrli bazaviy sinfni chaqirib olish. Aynan SimpleButton::Draw () metodini qayta yuklash tugmacha tashqi ramkasining ota sinfida sodir bo‘ladigan chiqarilishiga yo‘l qo‘ymaydi. Tabiiyki, metod kodini o‘zgartirish uchun uni TButtonControl bazaviy komponentasining dastlabki matni bo‘yicha o‘rganib chiqish lozim.

Endi izohlovchi nomga ega tugmachani yaratamiz. Buning uchun TButtonControl bazaviy sinfidan TextButton hosila sinfini qurish hamda Draw usulini, uning funksionalligini oshirgan holda qayta yuklash lozim.

Quyida Text sinfining title nomli obyekti TextButton konstruktori tomonidan yaratilishi, SimpleButton:-.Draw () metodida esa uning aks ettirilishi ko‘rsatilgan:

```
class Text { public:
Text(int x, int y, char* string) { };
 void Draw() { }
 };
class TextButton: public : TButtonControl {
Text* title;
public:
TextButton(int x, int y, char* title);
void Draw();
-TextButton() { }
};
TextButton::TextButton(int x, int y, char* caption):
TButtonControl(x, y) {
title = new Text(x, y, caption);
};
void TextButton::Draw () {
TextButton::Draw() ;
title->Draw() ;
}
```

Bo‘lim yakunida bazaviy va hosila sinflarini yaratish metodikasini bayon qilish bilan birga, dasturning C++fragmenti keltiriladi. Bu fragmentda ikkita oddiy geometrik obyekt — doira va silindrning sinflar tabaqalanishi e’lon qilingan.

Dastur shunday tuzilganki, bunda doiraning r — radiusi va silindrning h — balandligi o‘zgaruvchilarining ichki qiymatlari yaratilayotgan obyektlar parametrlarini aniqlashi kerak. Circle bazaviy

sinfi doirani modellashtiradi, Cylinder hosila sinfi esa silindrni modellashtiradi:

```
const double pi = 4 * atan(l);
class Circle { protected:
double r ;
public:
Circle (double rVal =0) : r(rVal) {}
void setRadius(double rVal) { r = rVal; }
double getRadiusO { return r; };
.double Area() { return pi*r*r; };
void showData() ;
};
class Cylinder : public Circle {
protected:
double h;
public:
Cylinder(double hVal = 0, double rVal = 0)
: getHeight(hVal), Circle(rVal) { };
void setHeight(double hVal) { h = hVal; }
double getHeight() { return h; };
double Area() { return 2*Circle::Area()+2*pi*r*h; };
 void showData() ;
void Circle::showData() {
cout << ″Doira radiusi = ″ << getRadius() << endl
<< ″Aylana maydoni = ″ << Area O << endl << endl;
};
void Cylinder::showData()
{
cout << ″Asos radiusi = ″<< getRadius()<<endl;
cout<< ″Silindr balandligi = ″<< getHeight()<< endl;
cout<<″Yuza maydoni = ″<<Area ()<< endl;
void main()
{
Circle circle(2) ;
Cylinder cylinder(10, 1);
circle.showData () ;
cylinder.showData() ;
}
```

Circle sinfining e'loni r ma'lumotlarining yagona a'zosi, konstruktor va qator metodlardan iborat. Obyektni yaratishda konstruktor r ma'lumotlar a'zosini doira radiusining boshlang'ich qiymati bilan nomlaydi (initsiallashtiradi). Konstruktorning yangi sintaksisini ko'rsatib o'tamiz: chaqirishda u bazaviy konstruktor sinfiga, shuningdek, ikki nuqtadan keyin ko'rsatilgan har qanday ma'lumotlar a'zosiga murojaat qilishi mumkin. Bizning misolimizda r ma'lumotlari a'zosi unga rVal parametri bilan murojaat qilish orqali «yaratiladi» va nolli qiymat bilan nomlanadi (initsiallashtiriladi).

setRadius usuli g ma'lumotlar a'zosi qiymatini belgilaydi, getRedius usuli esa uni qaytaradi. Area metodi doira maydonini qaytaradi. show Data metodi aylana radiusi va doira maydonining qiymatlarini chiqarib beradi.

Circle sinfining hosilasi deb e'lon qilingan Cylinder sinfi h — yagona ma'lumotlar a'zosi, konstruktor va qator metodlardan iborat. Bu sinf r ma'lumotlar a'zosini silindr asosi radiusini saqlash uchun setRadius va getRadius metodlarini meros qilib oladi. Obyektni yaratishda konstruktor r va h ma'lumotlar a'zolarini boshlang'ich qiymatlar bilan nomlaydi. Konstruktorning yangi sintaksisini ko'rsatib o'tamiz: bizning holatda h ma'lumotlar a'zosi hVal argumentining qiymati bilan nomlanadi (initsiallashtiriladi), r ma'lumotlar a'zosi esa rVal argumentiga ega bo'lgan bazaviy sinf konstruktorini chaqirish bilan nomlanadi.

setHeight funksiyasi h ma'lumotlar a'zosi qiymatini belgilaydi, getHeight esa qaytaradi. Circle::Area funksiyasi bazaviy sinfdan meros olingan funksiyani, silindr yuzasi maydonini qaytarish uchun, ortiqcha yuklaydi. ShowDate funksiyasi esa silindr asosining radiusi, balandligi va yuzasining maydoni qiymatlarini chiqarib beradi.

main funksiyasi Circle sinfiga mansub 2 radiusli circle aylanasini hamda balandligi 10, asosining radiusi 1 bo'lgan Cylinder sinfiga mansub silindrni yaratadi, keyin yaratilgan obyektlarning parametrlarini chiqarish uchun showData ga murojaat qiladi:

```
Aylana radiusi=2 Doira maydoni=12.566
Asos radiusi=1 silindr balandligi=10 Yuzasining maydoni=69.115
```

3.5. Polimorfizm

Polimorfizm yunoncha so'z bo'lib, ikkita o'zakdan — poly (ko'p) va morphos (shakl) dan iborat bo'lib, ko'p shakllilikni bildiradi. Polimorfizm — bu turdosh obyektlar (ya'ni bitta ajdod hosilasi bo'lgan

sinflarga mansub obyektlar) ning dastur bajarilish vaqtida vaziyatga qarab o‘zlarini turlicha tuta olish xususiyati. OMD doirasida dasturchi obyekt xulq-atvoriga faqat bilvosita ta’sir ko‘rsatishi, ya’ni dasturga kiritilayotgan metodlari o‘zgartirilishi hamda avlodlarga o‘z ajdodlarida yo‘q bo‘lgan o‘ziga xos xususiyatlarni baxsh etishi mumkin.

Metodni o‘zgartirish uchun uni avlodda ortiqcha yuklash kerak, ya’ni avlodda bitta nomdagi metodni e’lon qilish va unda kerakli xatti-harakatlarni ishga solish kerak. Natijada ajdod-obyekt va avlod-obyektda bitta nomdagi ikkita metod amal qiladi. Bunda ushbu metodlarning kodlari turlicha ishga tushiriladi va, demakki, obyektlarga turlicha xatti-harakat baxsh etadi. Masalan, geometrik shakllar turdosh sinflarining tabaqalanishida (nuqta, to‘g‘ri chiziq, kvadrat, to‘g‘riburchak, doira, ellips va h.k.) har bir sinf Draw metodiga ega bo‘lib, u ushbu shaklni chizib berish talabi qo‘yilgan voqea-hodisaga tegishli javob berilishi uchun mas’uldir.

Polimorfizm tufayli avlodlar bitta voqeaga o‘ziga xos tarzda munosabat bildirish uchun o‘z ajdodlarining umumiy metodlarini ortiqcha yuklashlari mumkin.

Virtual funksiyalar

OMD da polimorfizmga nafaqat yuqorida tavsifi berilgan vorislik va ajdod metodini ortiqcha yuklatish mexanizmi vositasida erishiladi, balki *virtuallash* vositasida ham erishiladiki, u ajdod funksiyalarga o‘z avlodlari funksiyalariga murojaat qilish imkonini beradi. Polimorfizm sinf arxitekturasi orqali ishga tushiriladi, biroq faqat a’zo-funksiyalar polimorf bo‘lishlari mumkin.

C++da polimorf funksiya bitta nomdagi ehtimoliy funksiyalardan biriga faqat bajarilish paytida, ya’ni unga sinfning aniq obyekti uzatilayotgan paytda bog‘lab qo‘yiladi. Boshqacha qilib aytganda, dastlabki dastur matnida funksiyaning chaqirilishi faqat taxminan belgilanadi, aynan qanday funksiya chaqirilayotgani aniq ko‘rsatilmaydi. Bu jarayon *kechikkan bog‘lanish* deb nom olgan. Navbatdagi misol oddiy a’zo-funksiyalarning polimorf bo‘lmagan xulq-atvori nimaga olib kelishi mumkinligini ko‘rsatadi:

```
class Parent { public:
double F1(double x) { return x*x; };
double F2(double x) { return Fl(x)/2; };
 class Child : public Parent { public:
double F1(double x) { return x*x*x; }
};
```

```
 void main() {
Child child;
cout << child.F2(3)<<endl;
}
```

Parent sinfi F1 va F2 a'zo-funksiyalarga ega, bunda F1 ni F2 chaqiradi. Parent sinfining hosilasi bo'lgan Child sinfi F2 funksiyasiga vorislik qiladi, biroq F1 funksiyasini oldindan belgilaydi. Kutilayotgan 13.5 natijasi o'rniga dastur 4.5 qiymatni chiqarib beradi. Gap shundaki, kompilyator child.F2(3) ifodasini meros qilib olingan Parent::F2 funksiyasi murojaatiga translyatsiya qilib yuboradi, bu funksiya esa o'z navbatida Child::F1 ni emas, Parent::F1 ni chaqiradi. Shunday bo'lganda edi, polimorf xulq-atvor qo'llab-quvvatlangan bo'lar edi.

C++kechikkan bog'lanishni funksiya bajarilish paytida aniqlaydi hamda funksiyalarni *virtuallash* vositasida ularning polimorf xulq-atvorini ta'minlaydi. Bazaviy va hosila sinflarda virtual funksiyalarni e'lon qilish sintaksisini umumlashtiradigan misolni ko'rib chiqamiz:

```
class className1 {
//Boshqa a'zo-funksiyalar
virtual return Type functionName(<parametrlar ro'yxati>);
}
class className2 : public className1 {
//Boshqa a'zo-funksiyalar
virtual return Type functionName(<>);
}
```

Parent va Child sinflari obyektlarida F1 funksiyasining polimorf xulq-atvorini ta'minlash uchun uni virtual deb e'lon qilish zarur. Quyida dasturning modifikatsiyalangan matni keltiriladi:

```
class Parent {
public:
virtual double F1(double x) { return x*x; }
double F2(double x) { return Fl(x)/2; }
};
 class Child : public Parent { public:
virtual double F1(double x) { return x*x*x; }
);
 void main() {
```

```
Child child;
cout ″ child.F2(3) ″ endl;
}
```

Mana endi dastur kutilayotgan 13.5 natijasini chiqarib beradi. Kompilyator child.F2(3) ifodasini meros qilib olingan Parent::F2 funksiyasi murojaatiga translyatsiya qilib yuboradi, bu funksiya esa, o‘z navbatida, Child::F1 avlodining qayta aniqlangan virtual funksiyasini chaqirib oladi.

Agar funksiya bazaviy sinfda virtual deb e’lon qilingan bo‘lsa, uni faqat hosila sinflarda qayta aniqlash mumkin, bunda parametrlar ro‘yxati avvalgidek qolishi zarur. Agar hosila sinfning virtual funksiyasi parametrlar ro‘yxatini o‘zgartirgan bo‘lsa, bu holda uning bazaviy sinfdagi (hamda uning barcha ajdodlaridagi) versiyasi kirib bo‘lmas bo‘lib qoladi. Boshida bunday vaziyat boshi berk ko‘chaga kirib qolgandek ko‘rinishi mumkin, amalda ortiqcha yuklanish mexanizmini qo‘llab-quvvatlamaydigan OMD tillarida shunday bo‘ladi ham. C++bu muammoni virtual funksiyalardan emas, balki xuddi shu nomli, faqat boshqa parametr ro‘yxatiga ega bo‘lgan ortiqcha yuklangan funksiyalardan foydalangan holda hal qiladi.

Virtual deb e’lon qilingan funksiya, hosila sinflarda **virtual** kalit-so‘z bilan e’lon qilingani yoki qilinmaganidan qat’i nazar, barcha hosila sinflarda virtual hisoblanadi.

Virtual funksiyalardan berilgan sinf obyektlarining o‘ziga xos xulq-atvorini ishga solish uchun foydalaning. Barcha metodlaringizni virtual deb e’lon qilmang, bu ularni chaqirishda qo‘shimcha hisoblash sarflariga olib keladi. Hammavaqt destruktorlarni virtual deb e’lon qiling. Bu sinflar tabaqalanishida obyektlarni yo‘q qilishda polimorf xulq-atvorni ta’minlaydi.

Polimorf obyekt-telefonning yaratilishi

Aytaylik, sizning boshliqlaringiz sizga obyekt-telefoningiz diskli, tugmachali yoki to‘lovli telefonlardan birini tanlab olib, emulyatsiya qila olishi kerak dedi. Boshqacha qilib aytganda, obyekt-telefon bitta qo‘ng‘iroq uchun tugmachali apparat sifatida, boshqa qo‘ng‘iroq uchun to‘lovli telefon sifatida va h.k. ishlashi mumkin edi. Ya’ni bir qo‘ng‘iroqdan ikkinchisiga sizning obyekt-telefoningiz o‘z shaklini o‘zgartirishi lozim bo‘ladi.

Turli sinflarga mansub bu telefonlarda faqat bitta farqlanuvchi funksiya mavjud — bu dial metodi. Polimorf obyektni yaratish uchun siz avval bazaviy sinf funksiyalarini, ularning prototiplari oldidan

virtual kalit-so‘zni qo‘ygan holda, aniqlaysiz. Bu bazaviy sinf funksiyalari hosila sinflar funksiyalaridan *virtualligi* bilan farqlanadi.

Keyin dasturda bazaviy sinf obyektiga ko‘rsatkich tuziladi. Sizning telefonga tuzilayotgan dasturingiz uchun siz *phone* bazaviy sinfiga ko‘rsatkich tuzasiz:

```
phone *poly_phone;
```

Obyekt shaklini o‘zgartirish uchun siz, quyida ko‘rsatilganidek, ushbu ko‘rsatkichga hosila sinf obyektining adresini berib qo‘yasiz:

```
poly_phone=(phone*)&home_phone;
```

Qiymat berish operatoridan keyin keladigan (phone*) belgilari turlarga keltirish operatori bo‘lib, bu operator C++ning kompilyatoriga hamma narsa joyida, bir turdagi o‘zgaruvchi (touch_tone) adresini boshqa turdagi o‘zgaruvchi (phone) ga berish zarurligini ma’lum qiladi. Dastur poly_phone obyekti ko‘rsatkichiga turli obyektlar adresini taqdim qilishi mumkin ekan, u holda bu obyekt ham o‘z shaklini o‘zgartirishi, demakki, polimorf bo‘lishi mumkin. Navbatdagi POLYMORPH.CPP dasturi bu metoddan obyekt-telefon yaratish uchun foydalanadi. Dastur ishga tushirilgach, poly_phone obyekti o‘z shaklini diskli telefondan tugmachalisiga, keyin esa to‘lovlisiga o‘zgartiradi:

```
#include <iostream.h>
#include <string.h>
class phone
{
public:
   virtual void dial(char *number) { cout <<″Raqam to‘plami ″ <<
number << endl; }
   void answer(void) { cout << ″Javobni kutish″ << endl; }
   void hangup(void) { cout << ″Qo‘ng‘iroq bajarildi — go‘shakni
qo‘yish″ << endl; }
   void ring(void) { cout << ″Qo‘ng‘iroq, qo‘ng‘iroq, qo‘ng‘iroq″
<< endl; }
   phone(char *number) { strcpy(phone::number, number); };
protected:
char number[13] ;
};
class touch_tone : phone
```

```
{
public:
  void dial(char * number) { cout << "Pik Pik Raqam to‘plami " << number << endl; }
touch_tone(char *number) : phone(number) { }
};
class pay_phone: phone
{
public:
  void dial(char *number) { cout << "Iltimos, to‘lang " << amount << " sentov" << endl; cout << "Raqam to‘plami " << number << endl; }
  pay_phone(char *number, int amount) : phone(number) { pay_phone::amount = amount; }
private:
  int amount;
};
void main(void)
{
pay_phone city_phone("702-555-1212", 25);
touch_tone home_phone("555-1212");
phone rotary("201-555-1212") ;
 // Obyekt diskli telefonga aylantirilsin
phone *poly_phone = &rotary;
poly_phone->dial("818-555-1212");
// Obyekt shakli tugmachali telefonga o‘zgartirilsin
poly_phone = (phone *) &home_phone;
poly_phone->dial("303-555-1212");
// Obyekt shakli to‘lovli telefonga o‘zgartirilsin
poly_phone = (phone *) &city_phone;
poly_phone->dial("212-555-1212");
}
```

Agar ushbu dastur kompilyatsiya qilinib ishga tushirilsa, displey ekranida quyidagi yozuv paydo bo‘ladi:

```
S:\> POLYMORPH <ENTER>
Raqam to‘plami 818-555-1212
Pik Pik Raqam to‘plami 303-555-1212
Iltimos 25 sent to‘lang
```

Raqam to‘plami 212-555-1212

Roly_phone obyekti dastur bajarilishi davomida o‘z shaklini o‘zgartirib turar ekan, u polimorf bo‘ladi.

Do‘stona funksiyalar

Do‘stona funksiyalar, garchi ular biror bir sinfga mansub bo‘lmasalar-da, tashqi sinf ma’lumotlarining barcha privat va himoyalangan a’zolariga kirish huquqiga ega bo‘ladilar. Do‘stona funksiyalarning e’lon qilinish sintaksisini qaytarilayotgan tur ko‘rsatkichi oldidan turgan friend kalit-so‘z yordamida ko‘rib chiqamiz:

```
class className
{
public: ~
className(); //Yashirish konstruktor//friend returnType friendFunction ning boshqa konstruktorlari(<parametrlar ro‘yxati>)
};
```

Agar oddiy a’zo-funksiyalar, sinf nusxasiga yashirish parametr — this ko‘rsatkichini uzatish hisobiga o‘z sinfining barcha ma’lumotlariga avtomatik tarzda kirish huquqiga ega bo‘lsa, do‘stona funksiyalar ushbu parametrning ochiq-oydin spetsifikatsiyasini talab qiladi.

Darhaqiqat, X sinfida e’lon qilingan F do‘stona funksiyasi bu sinfga mansub emas, demakki, x.F va xptr->F (bu yerda x — X sinfining nusxasi, xptr — uning ko‘rsatkichi) operatorlari tomonidan chaqirib olinolmaydi. Bu o‘rinda F(&x) yoki F(xptr) murojaatlari sintaktik jihatdan korrekt (to‘g‘ri) bo‘ladi.

Shunday qilib, do‘stona funksiyalar sinfning a’zo-funksiyalari vositasida ishga tushirilishi noqulay, qiyin va hatto mumkin bo‘lmagan masalalarni ham hal qilishlari mumkin.

Standart amallarni qo‘shimcha yuklash

Standart amallarni (masalan +) qo‘shimcha yuklash biror sinf bilan birga qo‘llashda mazmunini o‘zgartirishdan iboratdir.

Amallarni qo‘shimcha yuklash uchun quyidagi ta’rifdan foydalaniladi:

operator amal (operandlar ro‘yxati)

Quyidagi amallarni qo‘shimcha yuklash mumkin:

```
+ — * / % ^ & | ~ !
= < > += -= *= /= %= ^= &=
|= << >> >>= <<= == != <= >= &&
‖ ++ — [] () new delete
```

Bu amallar ustivorligi va ifodalar sintaksisini o‘zgartirishi mumkin emas.

Amal funksiyasi oddiy usulda chaqirilishi mumkin. Masalan:

```
void f(complex a, complex b)
 {
   complex c = a + b;      // qisqa yozuv
complex d = operator+(a,b); // oshkora chaqirish
 }
```

Binar amal bitta parametrga ega sinf usuli yoki ikki parametrga ega do‘stona funksiya sifatida ta‘riflanishi mumkin. Misollar:

```
class X {
 // do‘stlar
friend X operator-(X);   // unar minus
   friend X operator-(X,X);  // binar minus
   friend X operator-();     // xato: operandlar yo‘q
   friend X operator-(X,X,X); // xato: ternar amal
// sinf usuli (this parametrli)
   X* operator&(); // unar & (adres olish)
   X operator&(X); // binar & (Va amali)
   X operator&(X,X);  // xato: ternar amal
 };
```

Misol sifatida sanalar ustidan standart amallarni qo‘shimcha yuklashni ko‘ramiz:

```
static  int days[]={ 0,31,28,31,30,31,30,31,31,30,31,30,31};
class   dat {
        int  day,month,year;
public:
   void next();       // Keyingi kunni hisoblash usuli
   dat operator++();  //  ++ amali
   dat operator+(int);// «sana + butun son amali
friend dat operator+(int,dat);// hamma argumentlarni qiymat bo‘yicha oshkora //uzatish
              dat(int=0,int=0,int=0);
              dat(char *);
```

```
        ~dat();
    };

//——— Keyingi kunni hisoblash funksiyasi ——————————
void dat::next()
{
day++;
if (day > days[month])
    {
 if ((month==2) && (day==29) && (year%4==0)) return;
    day=1;  month++;
    if (month==13)
        {
        month=1; year++;
        }
    }
}

//——— Sana ustida amal ——————————————-
dat   dat::operator++()
{
dat x = *this;
dat::next();
return(x);
}

//——— «sana + butun son» amali ——————————————
dat   dat::operator+(int n)
{
dat   x;
x = *this;
while (n— !=0) x.next();
return(x);
}

//——— «butun son+sana» amali ——————————————-
dat   operator+(int n, dat p)
{
while (n— !=0) p.next();
return(p);
```

```
}

void main()
{
int i;
dat a, b(17,12,1990), c(12,7), d(3), e;
dat *p = new dat[10];
e = a++;
d = b+15;
for (i=0; i<10; i++) p[i] = p[i] + i;
delete[10] p;
}
```

Keyingi misolda () va [] amallarini satr uchun qo‘shimcha yuklashni ko‘ramiz:

```
//——— [] va () amallarini qo‘shimcha yuklash
#include <string.h>
class   string          // Satr
        {
        char    *Str;   // Simvollar dinamik massivi
        int     size;   // Satr uzunligi
public:
string  operator()(int,int); // Ostki satrni ajratish amali
char    operator[](int);     // Simvolni ajratish amali
int     operator[](char*);   // Ostki satrni izlash amali
        };
//——— Ostki satrni ajratish amali  ——————————-
string  string::operator()(int n1, int n2) {
string  tmp = *this;
delete  tmp.Str;
tmp.Str = new char[n2-n1+1];
strncpy(tmp.Str, Str+n1, n2-n1); }
```

C++ tilida quyidagi amallarni qo‘shimcha yuklash mumkin emas:

. sinf obyekti a’zosiga murojaat;
.* ko‘rsatkich orqali murojaat;
?: shartli amal;
:: ko‘rinish sohasini ko‘rsatuvchi amal;
sizeof hajmni hisoblash amali;

#	preprotsessor amali.

3.6. Shablonlar

Shablonlar (parametrlangan turlar) bog‘langan funksiyalar yoki sinflar oilasini tuzish imkonini beradi. Shablonni aniqlashning umumiy sintaksisi quyidagi ko‘rinishga ega:

temlate <shablonli turlar ro‘yxati>{e’lon qilish};

Funksiyalar shablonlari va sinflar shablonlari farqlanadi.

Funksiya shabloni ortiqcha yuklanayotgan funksiyalarni aniqlash namunasini beradi. Tartiblashtirishga yo‘l qo‘yadigan har qanday turdagi ikkita argumentdan kattarog‘ini qaytaradigan Tax(x, y) funksiyasi shablonini ko‘rib chiqamiz:

template <class T>T max(Tx, Ty){**return**(x>y)? x:y;}:

Bu holda <class T> shablonining argumenti tomonidan taqdim etilgan ma’lumotlar turi har qanday bo‘lishi mumkin. Undan dasturda foydalanishda kompilyator tax funksiyasi kodini bu funksiyaga uzatilayotgan parametrlarning faktik turiga muvofiq generatsiya qiladi:

```
int i;
Myclass a,b;
int i=max(i, 0);//argumentlar turi int myclass m=max(a, b);//
Myclass argumentlar turi
```

Faktik turlar kompilyatsiya paytida ma’lum bo‘lishi kerak. Shablonlarsiz max funksiyasini ko‘p martalab ortiqcha yuklashga to‘g‘ri kelar edi, ya’ni, garchi barcha funksiya versiyalarining kodlari bir xil bo‘lsa ham har bir qo‘llanayotgan tur uchun alohida ortiqcha yuklash kerak bo‘lar edi. C++standarti bu maqsad uchun **#define max (x, y) ((x>y)? x:y)** makrosidan foydalanishni qat’iyan tavsiya etmaydi, chunki C++tiliga oddiy S tili ustidan shunday afzalliklarni beradigan turlarni tekshirish mexanizmi blokirovka qilingan bo‘lishi mumkin. Shu narsa ayonki, max(x, u) funksiyasining vazifasi bir-biriga mos turlarni qiyoslash. Afsuski, makrosni qo‘llash bir-biriga mos kelmaydigan turlarning (masalan, int va struct) qiyoslanishiga yo‘l qo‘yadi:

```
//ikkita o‘zgaruvchining o‘rnini o‘zgartiradigan funksiya shabloni:
template<class T>//T — parametrlanayotgan tur nomi
```

```
void change(T*x, T*y)
{Tz=*x;  *x=*y;*y=z;}
```

Bu funksiya quyidagicha chaqirilishi mumkin:

```
long k=10,I=5;
chang(&k, &i);
```

Kompilyator:

void change(long*x, long*y){long z=*x; *x=*y; *y=z;} ta'rifini ifodalab beradi.

Funksiyalar shablonlari parametrlarining asosiy xususiyatlari:

1. Parametrlar nomlari shablonning butun ta'rifi bo'ylab unikal bo'lmog'i lozim.
2. Shablon parametrlarining ro'yxati bo'sh bo'la olmaydi.
3. Shablon parametrlari ro'yxatida har biri class so'zidan boshlanadigan bir nechta parametr bo'lishi mumkin.

Sinflar shabloni sinflar oilasini aniqlash namunasini beradi. Quyida bir o'lchamli ma'lumotlar massivi sinflarining generatori bo'lgan Vector shabloniga misol keltirilgan:

```
template <class T>class Vector
```

Ushbu sinfning turlashtirilgan elementlari ustida, elementlarning aniq turidan qat'i nazar, bir xil bazaviy operatsiyalar (kiritilsin, o'chirilsin, indekslansin va h.k.) bajariladi. Agar turga parametrdek muomala qilinsa, bu holda kompilyator berilgan tur elementlariga ega bo'lgan vektorlar sinflarini generatsiya qiladi.

Funksiyalar shablonlarida bo'lganidek,

class Vector<char*.(...}; sinflari shablonlarini ochiq-oydin qayta aniqlashga ruxsat etiladi, bunda Vector belgisi hammavaqt burchakli qavslar ichiga olingan ma'lumotlar turi bilan birgalikda kelishi kerak.

Navbatdagi dastur ikkita sinfni yaratishda array sinf shablonidan foydalanadi. Bu sinflarning biri int turining qiymatlari bilan, ikkinchisi float turining qiymatlari bilan ish ko'radi:

```
#include <iostream.h>
#include <stdlib.h>
template<class T, class T1> class array
{
public:
```

```
array(int size);
T1 sum(void);
T average_value(void);
void show_array(void);
int add_value(T);
private:
T *data;
int size;
int index;
};
template<class T, class T1> array<T, T1>::array(int size)
{
 data = new T[size];
   if (data == NULL)
{
    cerr<< "Xotira yetarli emas — dastur tugayapti" << endl;
exit(l);
}
 array::size = size;
array::index = 0;
}
template<class T, class T1> Tl array<T, Tl>::sum(void)
{
 T1 sum = 0;
for (int i = 0; i < index; i++) sum += data[i];
return(sum);
}
template<class T, class T1> T array<T, T1>::average_value(void)
{
Tl sum =0;
for (int i = 0; i < index; i++) sum += data[i];
return (sum / index);
}
template<class T, class T1> void array<T, T1>::show_array(void)
{
   for (int i = 0; i < index; i++) cout << data[i] << ' ';
cout << endl;
}
template<class T, class T1> int array<T, T1>::add_value(T value)
{
if (index == size)
```

```
return(-1); // Massiv to‘la
  else
{
data[index] = value;
index++;
return(0); // Omadli
}
}
void main(void)
{
 // 100 ta elementdan iborat massiv
array<int, long> numbers(100);
// 200 ta elementdan iborat massiv
array<float, float> values(200);
 int i;
for (i = 0; i < 50; i++) numbers.add_value(i);
numbers.show_array();
  cout << ″Sonlar summasi teng ″ << numbers.sum () << endl;
  cout << ″O‘rtacha qiymat teng ″ << numbers.average_value()
<< endl;
 for (i = 0; i < 100; i++) values.add_value(i * 100);
values.show_array();
  cout << ″ Sonlar summasi teng ″<< values.sum() << endl;
  cout << ″ O‘rtacha qiymat teng ″ << values.average_value() <<
endl;
}
```

3.7. Fayllar bilan ishlash

Oldindan belgilangan obyektlar va oqimlar

C++ da kiritish-chiqarish oqimlarining sinflari mavjud bo‘lib, ular kiritish-chiqarish standart kutubxonasi (stdio.h) ning obyektga mo‘ljallangan ekvivalenti (stream.h) dir. Ular quyidagicha:

ios	bazaviy oqimli sinf
streambuf	oqimlarning buferlanishi
istream	kiritish oqimlari
ostream	chiqarish oqimlari
iostream	ikkiga yo‘naltirilgan oqimlar
iostream_withassign	noaniq o‘zlashtirish operatsiyali oqim satrli kiritish oqimlari

istrstream	satrli chiqarish oqimlari
ostrstream	ikkiga yo‘naltirilgan satrli oqimlar
strstream	faylli kiritish oqimlari
	faylli chiqarish oqimlari
ifstream	ikkiga yo‘naltirilgan faylli oqimlar
ofstream	
fstream	

Standart oqimlar (istream, ostream, iostream) terminal bilan ishlash uchun xizmat qiladi.

Satrli oqimlar (istrstream, ostrstream, strstream) xotirada joylashtirilgan satrli buferlardan kiritish-chiqarish uchun xizmat qiladi.

Faylli oqimlar (ifstream, ofstream, fstream) fayllar bilan ishlash uchun xizmat qiladi.

Quyidagi obyekt-oqimlar dasturda main funksiyasini chaqirish oldidan avvaldan aniqlangan va ochilgan bo‘ladi:

```
extern istream cin;    //Klaviaturadan kiritish standart oqimi
extern ostream cout; //Ekranga chiqarish standart oqimi
extern ostream cerr; //Xatolar haqidagi xabarlarni chiqarish standart oqimi (ekran)
extern ostream cerr:  //Xatolar haqidagi xabarlarni chiqarishning buferlashtirilgan standart oqimi.
```

C++ da fayllar bilan ishlash sinflari

C++ da fayllar bilan ishlash fstream kutubxonasidagi biror bir sinflar yordamida amalga oshiriladi.

fstream kutubxonasi fayllarni o‘qib olish uchun javob beradigan ifstream sinfiga hamda faylga axborotning yozib olinishiga javob beradigan ofstream sinfiga ega.

Biror bir faylni yozish, o‘qish yoki ochish uchun ofstream turdagi yoki mos holda ifstream turdagi o‘zgaruvchini yaratish kerak. Bunday o‘zgaruvchini initsiallashda fayl nomi o‘zgaruvchi nomidan keyin qavs ichida berilgan belgilar massivi ko‘rinishida uzatiladi.

Masalan, S diskida joylashgan ′text.txt′ faylini ochish kerak. Buning uchun kodning quyidagi fragmenti qo‘llanadi:

```
ifstream ifI (″C:\text.txt″);        //o‘qish uchun
ofstream ofI(″C:\text.txt″):        //yozish uchun
```

Bu yerda ifI va ofI — o‘zgaruvchilar nomi bo‘lib, ular orqali fayl bilan ma’lumotlarni ayirboshlash amalga oshiriladi. Agar fayl ham

dasturning bajarilayotgan fayli joylashtirilgan papkada bo‘lsa, u holda faylning nomi to‘liq ko‘rsatilmasligi mumkin (faqat fayl nomi, unga borish yo‘lisiz). Bundan tashqari fayl nomini to‘g‘ridan-to‘g‘ri ko‘rsatish o‘rniga uning nomidan iborat belgilar massivlarini ko‘rsatish mumkin.

Masalan:

```
char s[20]=″C:\text.txt″;
ifstream ifI (s);
```

Axborotni faylga yozish uchun put buyrug‘idan foydalanish mumkin. Bu buyruq orqali standart turdagi yakka o‘zgaruvchi yoki biror bir belgilar massivi uzatiladi. Belgilar massivi uzatilgan holda ham massivdagi belgilar sonini uzatish kerak:

```
ofstream ofI (″C:\text.txt″);
char s[9]=″The text″;
ofI.put(s,9);
int i=100;
ofI.put(i);
```

put funksiyasini chaqirish o‘rniga ″<<″ operatoridan foydalanish mumkin:

```
ofstream ofI (″C:\text.txt″);
ofI<<″The text″;
int i=100;
ofI<<i;
```

Bu operatordan kodning bitta satrida turli turdagi qiymatlarni uzatgan holda ko‘p martalab foydalanish mumkin:

```
ofstream ofI (″C:\text.txt″);
char s[9]=″The text″;
int i=100;
ofI<<″The text″<<i<<s<<200;
```

Satr haqida gap ketganda, chiqarish satr oxiri belgisi, ya’ni ′\n′ paydo bo‘lishidan oldin amalga oshiriladi. Belgisiz turga ega bo‘lgan barcha o‘zgaruvchilar oldin belgilarga o‘zgartirib olinadi.

Axborotni fayldan o‘qib olish uchun ″>>″ operatoriga ekvivalent bo‘lgan get funksiyasi qo‘llanadi. put funksiyasi kabi get funksiyasi

ham har qanday o‘zgaruvchilarning standart turlari yoki/va belgilar massivlari bilan ishlay oladi. Shuningdek, get ga har jihatdan ekvivalent bo‘lgan getline funksiyasi mavjud: farqi faqat shundaki, getline funksiyasi satr oxiridan oxirgi belgini qaytarmaydi.

Fayl oxirini aniqlash

Fayl ichidagisini, fayl oxiri uchramaguncha, o‘qish dasturdagi oddiy fayl operatsiyasi hisoblanadi. Fayl oxirini aniqlash uchun dasturlar oqim obyektining eof funksiyasidan foydalanishlari mumkin. Agar fayl oxiri hali uchramagan bo‘lsa, bu funksiya 0 qiymatini qaytarib beradi, agar fayl oxiri uchrasa, 1 qiymatini qaytaradi. While siklidan foydalanib, dasturlar, fayl oxirini topmagunlaricha, quyida ko‘rsatilganidek, uning ichidagilarini uzluksiz o‘qishlari mumkin:

```
while (! Input_file.eof())
{
  //Operatorlar
}
```

Ushbu holda dastur, eof funksiyasi yolg‘on (0) ni qaytarguncha, siklni bajarishda davom etadi. Navbatdagi TEST_EOF.CPP dasturi BOOKINFO.DAT fayli ichidagisini, fayl oxiriga yetmaguncha, o‘qish uchun eof funksiyasidan foydalanadi:

```
#include <iostream.h>
#include <fstream.h>
void main (void)
{
 ifstream input_file(″BOOKINFO.DAT″);
char line[64];
   while (! input_file.eof())
{
  input_file.getline(line, sizeof(line));
  cout << line << endl;
}
}
```

Xuddi shunday, keyingi dastur — WORD_EOF.CPP fayl ichidagisini bitta so‘z bo‘yicha bir martada, fayl oxiri uchramaguncha, o‘qiydi:

```
#include <iostream.h>
#include <fstream.h>
```

```
void main(void)
{
 ifstream input_file(″BOOKINFO.DAT″);
char word[64] ;
   while (! input_file.eof())
{
input_file >> word;
cout << word << endl;
}
}
```

Va, nihoyat, keyingi dastur — CHAR_EOF.CPP fayl ichidagisini bitta belgi bo‘yicha bir martada get funksiyasidan foydalanib, fayl oxiri uchramaguncha, o‘qiydi:

```
#include <iostream.h>
#include <fstream.h>
void main(void)
{
ifstream input_file(″BOOKINFO.DAT″);
char letter;
   while (! input_file.eof())
{
letter = input_file.get();
cout << letter;
}
}
```

Fayl operatsiyalarini bajarishda xatolarni tekshirish

Hozirgacha taqdim etilgan dasturlarda ko‘zlanganidek, V/V fayl operatsiyalarini bajarishda xatolar sodir bo‘lmaydi. Afsuski, bunga hammavaqt ham erishib bo‘lmaydi. Masalan, agar siz kiritish uchun fayl ochayotgan bo‘lsangiz, dasturlar ushbu fayl mavjudligini tekshirib ko‘rishi kerak. Xuddi shunday, agar dastur ma’lumotlarni faylga yozayotgan bo‘lsa, operatsiya muvaffaqiyatli o‘tganiga ishonch hosil qilish kerak (masalan, diskda bo‘sh joyning yo‘qligi ma’lumotlarning yozib olinishiga to‘sqinlik qiladi). Xatolarni kuzatib borishda dasturlarga yordam berish uchun fayl obyektining fail funksiyasidan foydalanish mumkin. Agar fayl operatsiyasi jarayonida xatolar bo‘lmagan bo‘lsa, funksiya yolg‘on (0) ni qaytaradi. Biroq, agar xato uchrasa, fail funksiyasi haqiqatni qaytaradi. Masalan, agar dastur

fayl ochadigan bo‘lsa, u, xatoga yo‘l qo‘yilganini aniqlash uchun, fail funksiyasidan foydalanishi kerak. Bu quyida shunday ko‘rsatilgan:

```
ifstream input_file(″FILENAME.DAT″);
if (input_file.fail())
{
  cerr << ″Ochilish xatosi FILENAME.EXT″ << endl;
exit(1);
}
```

Shunday qilib, dasturlar o‘qish va yozish operatsiyalari muvaffaqiyatli kechganiga ishonch hosil qilishlari kerak. TEST_ALL.CPP dasturi turli xato vaziyatlarni tekshirish uchun fail funksiyasidan foydalanadi:

```
#include <iostream.h>
#include <fstream.h>
void main(void)
{
char line[256] ;
ifstream input_file(″BOOKINFO.DAT″) ;
   if (input_file.fail()) cerr << ″Ochilish xatosi BOOKINFO. DAT″
<< endl;
   else
{
      while ((! input_file.eof()) && (! input_file.fail()))
{
input_file.getline(line, sizeof(line)) ;
  if (! input_file.fail()) cout << line << endl;
}
}
}
```

Faylning kerak bo‘lmay qolganda berkitilishi

Dasturni tugallash uchun operatsiya tizimi o‘zi ochgan fayllarni berkitadi. Biroq, odatga ko‘ra, agar dasturga fayl kerak bo‘lmay qolsa, uni berkitishi kerak. Faylni berkitish uchun dastur, quyida ko‘rsatilganidek, dastur close funksiyasidan foydalanishi kerak:

```
input_file.close ();
```

Faylni yopayotganingizda dastur ushbu faylga yozib olgan barcha ma'lumotlar diskka tashlanadi va ushbu fayl uchun katalogdagi yozuv yangilanadi.

O'qish va yozish operatsiyalarining bajarilishi

Hozirga qadar gap borayotgan dasturlar belgili satrlar ustida operatsiyalar bajarar edi. Dasturlaringiz murakkablashgan sari, ehtimol, sizga massivlar va tuzilmalarni o'qish va yozish kerak bo'lib qolar. Buning uchun dasturlar read va write funksiyalaridan foydalanishlari mumkin. read va write funksiyalaridan foydalanishda ma'lumotlar o'qiladigan yoki yozib olinadigan ma'lumotlar buferini, shuningdek, buferning baytlarda o'lchanadigan uzunligini ko'rsatish lozim. Bu quyida ko'rsatilganidek amalga oshiriladi:

```
input_file.read(buffer, sizeof(buffer));
output_file write(buffer, sizeof(buffer));
```

Masalan, STRU_OUT.CPP dasturi tuzilma ichidagisini EMPLOYEE.DAT fayliga chiqarish uchun write funksiyasidan foydalanadi:

```
#include <iostream.h>
#include <fstream.h>
void main(void)
{
  struct employee
{
 char name[64];
int age;
float salary;
  } worker = { "Djon Doy", 33, 25000.0 };
ofstream emp_file("EMPLOYEE.DAT") ;
 emp_file.write((char *) &worker, sizeof(employee));
}
```

Odatda write funksiyasi belgilar satriga ko'rsatkich oladi. (char*) belgilari turlarga keltirish operatori bo'lib, bu operator siz ko'rsatkichni boshqa turga uzatayotganingiz haqida kompilyatorga axborot beradi. Xuddi shunday tarzda STRU_IN.CPP dasturi read metodidan xizmatchi haqidagi axborotni fayldan o'qib olish uchun foydalanadi:

```
#include <iostream.h>
```

```
#include <fstream.h>
void main(void)
{
  struct employee
{
char name [6 4] ;
int age;
float salary;
  } worker = { ″Djon Doy″, 33, 25000.0 };
 ifstream emp_file(″EMPLOYEE.DAT″);
emp_file.read((char *) &worker, sizeof(employee));
cout << worker.name << endl;
cout << worker.age << endl;
cout << worker.salary << endl;
}
```

3.8. Istisnolar

C++tili OMD doirasida istisnolarga xizmat ko‘rsatish standartini belgilab beradi. C++Builder Vizual Komponentalar Kutubxonasidan foydalanishda istisnolar (xatolar) ga ishlov berish uchun maxsus mexanizmlarni ko‘zda tutadi. C++Builder, shuningdek, operatsiya tizimining istisnolariga ishlov berishni hamda ilova ishining tugallanish modelini qo‘llab-quvvatlaydi.

Dastur o‘zining ishlab chiqilishida ko‘zda tutilmagan normal bo‘lmagan vaziyatga duch kelganda, boshqaruvni ushbu muammoni hal qilishga qodir bo‘lgan dasturning boshqa qismiga berish mumkin hamda yo dasturni bajarishni davom ettirish, yoki ishni tugallash kerak. *Istisnolarni joydan joyga tashlab berish (excpletion throwing)* dasturning normal bajarilishiga to‘sqinlik qiladigan sabablarning tashxisi uchun foydali bo‘lishi mumkin bo‘lgan axborotni tashlab berish nuqtasida to‘plash imkonini beradi. Siz dastur tugallanishi oldidan zarur xatti-harakatlarni bajaradigan *istisnolarga ishlov bergich (exception handler)* ni aniqlashingiz mumkin. Dastur ichida yuzaga keladigan sinxron istisnolar deb nomlanuvchi istisnolarga xizmat ko‘rsatiladi. Ctrl+C klavishalarini bosish kabi tashqi holatlar istisno hisoblanmaydi.

Istisnolarni generatsiya qila oladigan kod bloki try kalit-so‘z bilan boshlanadi va shakldor qavslar ichiga olinadi. Agar try bloki ushbu blok ichida istisnoni topib olsa, dasturiy uzilish sodir bo‘ladi hamda quyidagi xatti-harakatlar ketma-ketligi bajariladi:

1.Dastur istisnoga ishlov bergichning to‘g‘ri keladiganini qidiradi.

2.Agar ishlov bergich topilsa, stek tozalanadi va boshqaruv istisnolarga ishlov bergichga uzatiladi.

3.Agar ishlov bergich topilmagan bo‘lsa, ilovani tugatish uchun terminate funksiyasi chaqiriladi.

Yuzaga kelgan istisnoga ishlov beruvchi kod bloki catch kalit-so‘z bilan boshlanadi va shakldor qavs ichiga olinadi. Istisnoga ishlov bergichning kamida bitta kod bloki bevosita try blokining ortidan kelishi kerak. Dastur generatsiya qilishi mumkin bo‘lgan har bir istisno uchun o‘z ishlov bergichi ko‘zda tutilgan bo‘lishi kerak. Istisnolarga ishlov bergichlar navbatma-navbat ko‘rib chiqiladi hamda turi bo‘yicha catch operatoridagi argument (dalil) turiga to‘g‘ri keladigan istisnoga ishlov bergich tanlab olinadi. Ishlov bergich tanasida goto operatorlari bo‘lmagan taqdirda, berilgan try bloki istisnolariga ishlov bergichning oxirgisidan keyin kelgan nuqtadan boshlab dasturning bajarilishi yana davom etadi. Istisnolarga ishlov berishning umumlashma sxemasi quyidagi ko‘rinishga ega:

```
Try{
//Istisnoni generatsiya qilishi mumkin bo‘lgan har qanday kod
}
catch(T X) |
{
//T turdagi X istisnolarga ishlov bergich, istisno |
//avval kelgan try blokining ichida avval generatsiya qilingan bo‘lishi mumkin // Avval kelgan try catch(...) blokining boshqa istisnolariga ishlov bergichlar
//Avval kelgan try blokining har qanday istisnosiga ishlov bergichlar
}
```

Istisno yuzaga kelganda, throw operatoridagi <ifoda> nom berish ifodasi muvaqqat obyektni nomlaydi (initsiallashtiradi). Bunda muvaqqat obyektning turi ifoda argumenti (dalili) ning turiga mos keladi. Ushbu obyektning boshqa nusxalari, masalan, istisno obyektidan nusxa ko‘chirish konstruktori yordamida generatsiya qilinishi mumkin.

Garchi operatsiya tizimining o‘zi bajarilayotgan dasturni ko‘zda tutilmagan holatlardan «chiqarib olish»ga harakat qilsa hamki, xatolarga ishlov berishning qabul qilingan standart metodikasiga amal qilish — ishonchli ilovalarni qurishning kafolatlaridan biridir.

Misol:

Ikkita suzuvchi sonlarning bo‘linishidan hosil bo‘lgan bo‘linmaning interaktiv hisoblagichidan iborat dasturni ko‘rib chiqamiz. Bu dastur cheksiz sikl (davr) da bo‘linuvchi va bo‘luvchi qiymatlarini so‘rab oladi, favqulodda vaziyat yuzaga kelganda esa, istisnoni keltirib chiqaradi. Istisnoni taqdim etayotgan MyDivideByZeroError sinfi istisnolarni boshqarish tizimi bilan samarali o‘zaro aloqa uchun barcha zarur elementlar to‘plamiga ega.

```
#include <iostream.h>
#include <string.h>
#define YESMESS ″Biz davom etamiz.″
#define NOMESS  ″Biz tugallaymiz.″
class MyDivideByZeroError
{
char *MyErrorMessage;
public:
char ContinueKey;
MyDivideByZeroError(): MyErrorMessage(NULL)
{
char YesKey;
 cout << ″Nolga bo‘lish qayd etildi.″ << endl;
 cout << ″Tezkor choralar ko‘rilsinmi? (Y/N) >> ″;
 cin >> YesKey;
 if ( YesKey == ′Y′ || YesKey == ′y′ )
 {
  ContinueKey = 1;
  MyErrorMessage = strdup(YESMESS);
 }
 else
 {
  ContinueKey = 0;
  MyErrorMessage = strdup(NOMESS);
 }
}
MyDivideByZeroError(const MyDivideByZeroError& CopyVal)
{
  ContinueKey = CopyVal.ContinueKey;
  MyErrorMessage = strdup(CopyVal.MyErrorMessage);
```

```
}
~MyDivideByZeroError()
{
 if (MyErrorMessage) delete(MyErrorMessage);
}
void PrintMessage()
{
 cout << MyErrorMessage << endl;
}
};
float Dividor(float, float) throw(MyDivideByZeroError);
void main()
{
 float MyVal1, MyVal2;
 for (;;)
 {
//__ Nazorat ostidagi blokning boshi _____________________.
  try
  {
   cout << "=========================" << endl;
  cout << "MyVal1 >> ";
  cin >> MyVal1;
  cout << "MyVal2 >> ";
  cin >> MyVal2;
 cout << "Hisoblaymiz... " << Dividor(MyVal1, MyVal2) << endl;
  cout << "Uddaladik! ";
  }
  catch (MyDivideByZeroError MyExcept)
  {
MyExcept.PrintMessage();
if (MyExcept.ContinueKey == 0)
{
 cout << "Xatolar bilan kurashish jonga tegdi! Ketdik." << endl;
 break;
}
  }
//__ Nazoratdagi blokdan tashqarida _____________________.
  cout << "Blokdan tashqaridamiz. Davom etamiz..." << endl;
 }
}
```

```
float Dividor(float Val1, float Val2) throw(MyDivideByZeroError)
{
 if (Val2 == 0.0) throw MyDivideByZeroError();
 return Val1/Val2;
}
```

3.9. C++ tilining yangi imkoniyatlari

C++ Builder, nafaqat ANSI C++ standarti kiritayotgan yangiliklarni qo‘llab-quvvatlaydi, balki tilni yangi imkoniyatlar bilan boyitadi. Shuni tushunib olish muhimki, tilni kengaytirish hech qachon quruq maqsad bo‘lib qolmagan va siz hali-hamon standart C++ doirasida yozilgan matnlarni kompilyatsiya qila olasiz. Biroq ilovalarni tez ishlab chiqish texnologiyasi (RAD) uchun C++ Builder taqdim etgan imtiyozlardan to‘liq foydalanishda kiritilgan til kengaytirishlarni qabul qilishingizga to‘g‘ri keladi.

Kengaytirishlarning ayrimlari (masalan, _classid) ni C++ Builder asosan ichki foydalanish uchun rezervlaydi. Boshqa kengaytirishlar (_int8,_int6 va h.k.) ochiq-oydin tushunarli bo‘lib turibdi, shuning uchun bu yerda ular ko‘rib chiqilmaydi. Bizning diqqatimiz C++ ning eng ahamiyatli kengaytirishlariga qaratiladi. Ular asosan tarkibli sinflarga mansub bo‘lib, kitob matnida ham, C++ Builder muhitida ishlab chiqilayotgan ilovalaringizda ham muttasil uchrab turadi.

Komponentlar (tarkibiy qismlar)

Komponentlar ko‘p o‘rinda, C++ standart sinflariga qaraganda, yuqoriroq darajadagi inkapsulatsiyalashga erishadilar. Buni tugmachaga ega bo‘lgan dialogni ishlab chiqish kabi oddiy misolda ko‘rib chiqamiz. Windows uchun namunaviy C++ dasturida tugmachani «sichqoncha» bilan bosish natijasida WM_LBUTTON DOWN xabarining generatsiyasi sodir bo‘ladi. Bu xabarni dastur yo switch operatorida, yoki chaqiriqlar jadvali (RESPONCE_TABLE) ning tegishli satrida «tutib olish»i, keyin esa ushbu xabarga javob protsedurasiga uzatishi kerak.

C++ Builder o‘zlashtirilishi qiyin bo‘lgan bu kabi dasturlash o‘yinlariga chek qo‘ydi. Komponent tugmachasi avvaldanoq unga OnClick voqeasi bilan bosishga javob beradigan qilib dasturlangan. Bu o‘rinda talab qilinayotgan narsa — tayyor metodni tanlab olish (yoki o‘zinikini yozish) hamda obyektlar Inspektori yordamida berilgan voqea-hodisaga ishlov bergichga kiritish lozim.

Komponentli sinflarni e'lon qilish

C++ Builder tarkibiga kiradigan Vizual Komponentalar Kutubxonasi — VCL sinflarining ilgarilovchi e'lonlari _declspec modifikatoridan foydalanadi:

_declspec(<spetsifikator>)

Bu kalit-so'z nafaqat bevosita modifikatsiyalanayotgan e'lon oldidan, balki e'lonlar ro'yxatining to'g'ri kelgan yerida paydo bo'lishi mumkin, bunda spetsifikator quyidagi qiymatlardan birini qabul qiladi:

delphiclass — u TObject sinfiga tegishli VCL ning bevosita yoki bilvosita hosilalarining ilgarilovchi e'loni uchun qo'llanadi. U VCL ning RTTI, konstruktorlar, destruktor va istisnolar bilan muomalasida muvofiqlik qoidalarini belgilaydi.

delphireturn — u Currency, AnsiString, Variant, TDateTime va Set sinflariga tegishli VCL ning bevosita yoki bilvosita hosilalarining ilgarilovchi e'loni uchun qo'llanadi. U VCL ning parametrlar va a'zo — funksiyalarning qaytarilayotgan qiymatlari bilan muomalasida muvofiqlik qoidalarini belgilaydi.

Pascalimplementation tarkibli sinf obyektli Paskal tilida ishga tushirilganini ko'rsatadi.

VCL sinf quyidagi cheklanishlarga ega:

■ Virtual bazaviy sinflarga vorislik qilish man etilgan.

■ Tarkibli sinflarning o'zlari vorislik uchun bazaviy sinf sifatida xizmat qila olmaydi.

■ Tarkibli obyektlar uyumning dinamik xotirasida new operatori yordamida yaratiladi.

Xususiyatlarni e'lon qilish

C++Builder tarkibli sinflar xususiyatlarini identifikatsiya qilish uchun _property modifikatoridan foydalanadi. Xususiyatni tavsiflash sintaksisi quyidagi ko'rinishga ega:

property<xususiyat turi> <xususiyat nomi> = {<atributlar ro'yxati>};

Bu yerda atributlar ro'yxati quyidagi xususiyatlar atributlarining sanog'iga ega:

write=<ma'lumotlar a'zosi yoki yozuv metodi> ma'lumotlar a'zosiga qiymat berish usulini aniqlaydi;

read=<ma'lumotlar a'zosi yoki o'qish metodi> ma'lumotlar a'zosining qiymatini olish usulini aniqlaydi;

default=<bul konstantasi> .dim kengayishli shaklga ega bo'lgan yashirin xususiyatlar qiymatini saqlashga ruxsat beradi yoki man etadi;

stored=<bul konstantasi yoki funksiya> .dfm. kengayishli shaklga ega bo'lgan faylda xususiyat qiymatini saqlash usulini aniqlaydi.

C++Builder ilovani loyihalash bosqichida obyektlar Inspektori tomonidan aks ettiriladigan komponentalar xususiyatlarini spetsifikatsiyalash uchun _published modifikatoridan foydalanadi. Agar komponentaning ishlab chiquvchisi biror bir xususiyat qiymatini modifikatsiyalashga ruxsat berishni xohlab qolsa, bu xususiyat _published sifatida e'lon qilinmaydi. Ushbu kalit-so'z bilan aniqlanayotgan ko'rimlilik qoidalari public sifatida e'lon qilingan ma'lumotlar a'zolari, metodlar va xususiyatlarning ko'rimlilik qoidalaridan farq qilmaydi. Yagona farq shundaki, dasturning ishlash paytida obyektlar Inspektoriga RTTI axboroti uzatiladi.

Voqealar ishlatgichlarining e'lonlari

C++Builder voqealar ishlatgichlari funksiyalarining e'loni uchun _closure modifikatoridan foydalanadilar:

<tur>(_closure*<name>)(<parametrlar ro'yxati>)

Bu kalit-so'z funksiya ko'rsatkichini name nomi bilan aniqlaydi. Oddiy funksiyaning 4 baytli manzil ko'rsatkichidan farqli o'laroq (bu ko'rsatkich CS:IP kod registrlariga uzatiladi), 8 baytli _closure yana yashirin parametrni ham uzatadi (joriy sinf ekzemplyariga this o'zgaruvchan ko'rsatkichi).

8 baytli ko'rsatkichlarning kiritilishi nafaqat aniqlangan sinfning biror bir funksiyasini chaqirib olish imkonini beradi, balki ushbu sinfning aniqlangan ekzemplyaridagi funksiyaga murojaat qilish imkonini ham beradi. Bu qobiliyat obyektli Paskaldan o'zlashtirilgan edi, **_closure** esa Vizual Komponentalar Kutubxonasidagi voqealar mexanizmini ishga tushirishda havodek zarur bo'lib qoldi.

Funksiyalarning tez chaqirilishi

Parametrlari protsessorli registrlar orqali uzatiladigan funksiyalarni e'lon qilishda **_fastcall** modifikatori qo'llanadi:

<qaytarilayotgan tur>**_fastcall**<name>(<parametrlar ro‘yxati^

Bu kalit-so‘z name nomli dastlabki uchta turlashtirilgan parametr (ro‘yxat bo‘yicha chapdan o‘ngga) stek orqali emas, balki AX, BX va DX protsessorli registrlar orqali uzatilishini aniqlaydi. Agar parametr qiymati registrga sig‘masa, ya’ni parametr orqali suzuvchi nuqtali sonlarni, tuzilmalar va funksiyalarni uzatishda u qo‘llanmaydi.

Xulosa qilib aytganda, funksiyalarning tez chaqirilishi C++Builder kompilyatoriningina vazifasiga kirmaydi. Voqealarga ishlov berish funksiyalarini e’lon qilishda **_fastcall** ning qo‘llanishiga alohida e’tibor berish kerak. Bu voqealarni C++Builder avtomatik tarzda generatsiya qiladi.

Nomlar fazosi

Oddiy ilovalarning ko‘pi dastlabki dastur matniga ega bo‘lgan bir nechta fayldan iborat. Bu fayllar dasturchilar guruhi tomonidan yaratilishi va xizmat ko‘rsatilishi mumkin. Pirovard natijada barcha fayllar birga to‘planadi va tayyor ilovani yig‘ishdan iborat bo‘lgan so‘nggi protseduradan o‘tadi. An’anaviy tarzda qabul qilinishicha, biror bir lokal soha (funksiya, sinf tanasi yoki translyatsiya moduli) ga kiritilmagan barcha nomlar umumiy global ismlarni bo‘lib olishadi. Shuning uchun, agar ayrim modullarni yig‘ish jarayonida nomlar takroran aniqlangani ayon bo‘lib qolsa, bu holda har bir nomni qandaydir yo‘l bilan farqlash zarurligini talab qiladi. C++da bu muammoning yechilishi nomlar fazosi (namespace) mexanizmi zimmasiga yuklatilgan.

Bu mexanizm ilovani bir necha tarmoq tizimlar (tizimchalar) ga bo‘lib tashlash imkonini beradi, bunda har bir tarmoq tizim nomlarni tanlashda erkin ish tutadi hamda uning muallifi xuddi shunday ismlardan biron boshqa kimsa foydalanishi mumkinligiga qayg‘urmasa ham bo‘ladi. Har bir tarmoq tizim global nomlar umumiy fazosida o‘zining paydo bo‘lganini namespace kalit-so‘zdan keyin kelgan unikal identifikator yordamida identifikatsiya qiladi:

namespace<identifikator> {[<e’lon qilish>]}

Identifikatsiya qilingan nomlar fazosi elementlariga kirishning uchta usuli mavjud:

■ Konkret elementga ochiq-oydin kirish kvalifikatsiyasi:
ALPHA :: vart;//ALPHA dagi o‘zgaruvchiga kirish;
■ Barcha elementlarga kirish:

using namespace::ALPHA;//ALPHA dagi barcha nomlarga kirish;

■ Nomlarning lokal fazosida yangi identifikatorning e'lon qilinishi:

using :: new_name;//identifikatorning qo'shilishi

Ochiq-oydin e'lonlar

Odatda bitta parametrli konstruktor e'lon qilingan sinf obyektlariga turlari avtomatik tarzda (yashirin holda) o'z sinfi turiga qayta o'zgaradigan qiymatlarni berish mumkin. Konstruktorni e'lon qilishda **explicit** konstruktoridan foydalanish mumkin:

explicit<konstruktorni e'lon qilish>

Bu holda berilgan sinf konstruktorlarini explicit kalit-so'z bilan e'lon qilishda sinfning barcha obyektlariga faqat shunday qiymatlarni berish mumkinki, bu qiymatlar turlari o'z sinfi turiga ochiq-oydin qayta o'zgaradigan bo'lishi kerak:

```
class X
public:
explicit X(int);
explicit X(const char*, int = 0);
};
void f(X arg)
(
X a = X (1) ;
X b = X(″satr″,0);
a = X(2);
}
```

Konstruktorlarning ochiq-oydin e'lonlari shuni talab qiladiki, nom berish operatorlaridagi qiymatlar qaysi sinfiy tur obyektlariga berilgan bo'lsa, ular xuddi shu sinfiy turga qayta o'zgartirilishini talab qiladi.

O'zgaruvchan e'lonlar

Fon masalasi, uzish ishlatgichi yoki kiritish-chiqarish porti tomonidan o'zgartirilishi mumkin bo'lgan o'zgaruvchini e'lon qilishda **volatile** modifikatori qo'llanadi:

volatile<tur><obyekt nomi>;

C++da volatile kalit-so'zning qo'llanishi sinflar va a'zo-funksiyalarga ham tegishlidir. Bu kalit-so'z ko'rsatilgan obyekt qiymatiga nisbatan taxminlar qilishni kompilyatorga taqiqlaydi, chunki

bunday qilinsa, ushbu obyektni o‘z ichiga olgan ifodalarni hisoblashda, uning qiymati har bir daqiqada o‘zgarib ketishi mumkin. Bundan tashqari o‘zgarib turadigan o‘zgaruvchi register modifikatori bilan e’lon qilinishi mumkin emas. Listing 3.15 ticks o‘zgaruvchisini vaqtli uzilishlar qayta ishlagichi modifikatsiya qiladigan taymerni ishga tushirishga misol bo‘la oladi.

```
volatile int ticks;
void timer( ) // Taymer funksiyasini e’lon qilish
ticks++;
void wait (int interval)
ticks = 0;
while (ticks < interval); // Kutish sikli
}
```

Aytaylik, *uzilishni qayta ishlatgichi— timer* real vaqt soatidagi apparat uzilishi bilan tegishli tarzda assotsiatsiya qilindi. ticks o‘zgaruvchisining qiymati ushbu qiymat parametri tomonidan berilgan vaqt intervaliga teng kelmaguncha wait protsedurasi kutish siklini ishlataveradi. C++kompilyatori sikl ichidagi har bir qiyoslash oldidan volatile ticks o‘zgaruvchisining qiymatini, sikl ichidagi o‘zgaruvchining qiymati o‘zgarmaganiga qaramay, ortiqcha yuklashi lozim. Ayrim optimallashtiruvchi kompilyatorlar bunday «Xavfli» xatoga yo‘l qo‘yishlari mumkin.

Hatto konstantali ifodaga kirganiga qaramay o‘zgartirilishi mumkin bo‘lgan o‘zgaruvchan o‘zgaruvchining boshqa bir turi **mutable** modifikatori yordamida e’lon qilinadi:

mutable<o‘zgaruvchining nomi>;

mutable kalit-so‘zning vazifasi shundan iboratki, u biror bir sinf ma’lumotlari a’zolarini spetsifikatsiya qiladi, bunda ushbu ma’lumotlar a’zolari mana shu sinfning konstantali funksiyalari tomonidan modifikatsiya qilinishi mumkin bo‘lishi kerak. Ma’lumotlar a’zosi sount ni F1 konstantali funksiya modifikatsiya qiladigan misolni ko‘rib chiqaylik:

```
class A{
public: mutable int count; int F1 (int p=0)const// F1 funksiyasini e’lon qilish
count=p++return count;//PI count ni qaytarib beradi
}
void main(){
```

```
A, a;
cout<<a.F1(3)<<endl; //main 4 qiymatini chiqarib beradi
```

RTTI turlarining identifikatsiyasi

RTTI (Run-Tame Type Identification) dasturini bajarishda turlarning identifikatsiyasi sizga o‘tkaziladigan dasturni yozish imkonini beradi. Bunda, agar bajarilish vaqtida dasturda ushbu obyekt ko‘rsatkichigagina kirish huquqi bo‘lgan taqdirda ham dastur obyektning faktik turini aniqlashga qodir bo‘ladi. Bu, masalan, virtual bazaviy sinf ko‘rsatkichini ushbu sinfga mansub faktik obyektning hosila turi ko‘rsatkichiga qayta o‘zgartirish imkonini beradi. Shunday qilib, turlar faqat statik tarzda — kompilyatsiya fazasidagina emas, balki dinamik tarzda — bajarilish jarayonida ham qayta o‘zgartirilishi mumkin. Ko‘rsatkichni berilgan turga dinamik qayta o‘zgartirish **dynamic_cast** operatori yordamida amalga oshiriladi.

RTTI mexanizmi ham obyekt biror bir aniq turga egami yoki ikkita obyektning ikkalasi ham bitta turga tegishli ekanini tekshirib ko‘rish imkonini beradi. typeid operatori argument (dalil) ning faktik turini aniqlaydi hamda ko‘rsatkichni ushbu turni tavsiflaydigan typeinto sinfi obyektiga qaytaradi. Bajarilish paytida RTTI ni obyektlar Inspektoriga qaytarar ekan, C++Builder unga ushbu sinf xususiyatlari va a’zolarining turlari haqida axborot beradi.

4. C++BUILDER ASOSLARI

4.1. Dastlabki tanishuv

Ishlab chiqishning integratsiyalashgan muhiti Komponentalar Palitrasini birlashtiradi. Shakllar Muharriri, Kod Muharriri, Obyektlar Noziri, Obyektlar Xazinasi — bular hammasi kod va zaxiralar ustidan to‘liq nazoratni ta’minlovchi dasturiy ilovalarni tez ishlab chiqish instrumentlari (4.1-rasm).

■ ***Komponentalar Palitrasi*** ilovalarni qurishda taklif qilinadigan 100 dan ortiq takroran qo‘llanadigan komponentalardan iborat.

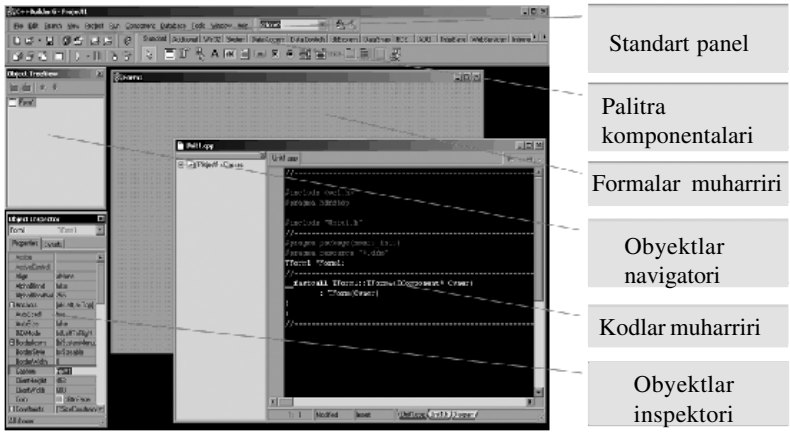
■ ***Shakllar Muharriri*** dasturning foydalanuvchi bilan interfeysini yaratish uchun mo‘ljallangan.

■ ***Kod Muharriri*** dastur matnini, xususan, voqealarga ishlov berish funksiyalarini yozish uchun mo‘ljallangan.

■ ***Obyektlar Noziri*** qotib qolgan chigal dasturlash zaruratisiz obyektlar xususiyatlarini vizual o‘rnatish imkonini beradi hamda shunday voqealarni

o‘z ichiga oladiki, bu voqealarni ularning paydo bo‘lishiga nisbatan obyektlar reaksiyasi kodlari bilan bog‘lash mumkin bo‘ladi.

■ ***Obyektlar Xazinasi*** ma’lumotlarning shakl va modullari kabi obyektlarga ega bo‘lib, ular ishlab chiqishda muvaqqat sarflarni kamaytirish maqsadida ko‘plab ilovalar bilan bo‘linadi.



*4.1-rasm.*Ishlab chiqish muhitining tuzilishi.

C++ Builder ilovalarni qurishning vizual metodikasini Komponentalar Palitrasidan kerakli boshqarish elementlarini tanlab olish vositasida joriy etadi. Har bir komponenta (masalan, tugmacha) bilan ushbu komponenta turini va xulq-atvorini o‘zgartiradigan xususiyatlar bog‘liq bo‘ladi. Har qanday komponenta ushbu komponentaning turli xildagi ta’sirlarga reaksiyasini (munosabatini) aniqlab beradigan voqealar seriyasini keltirib chiqarishi mumkin. Bundan keyin => belgilari siz C++Builder muhitida amalga oshiradigan xatti-harakatlarni bildiradi.

=>C++ Builder ni chaqiring va bosh menudagi Fale | New Arrlication buyrug‘i bo‘yicha yangi ilovalar ustida ishlashni boshlang.

=>sichqonchani Komponentalar Palitrasining qo‘shimcha ilovalari ustida bosib, foydalanuvchi ish ko‘radigan dastur interfeysi elementlarining mavjud assortimentini ko‘rib chiqing.

Palitraning bir qo‘shimcha ilovasidan ikkinchisiga o‘tib, kirish mumkin bo‘lgan komponentalar to‘plami o‘zgarayotganining guvohi bo‘lishimiz mumkin. Sichqoncha kursori komponentalar belgisi ustida to‘xtaganda, aytib turish nomi paydo bo‘ladi. Agar F1 klavishasini bossak, tizimning ma’lumotnomalar xizmati tanlab olingan komponenta haqida to‘liq ma’lumot chiqarib beradi.

Vizual loyihalash

Bizning birinchi ilovamiz bolalarning «O‘nta negr bolasi» sanoq she’rini generatsiya qiladi. Dastlabki versiyada faqat uchta obyekt kerak bo‘ladi: ro‘yxat, tahrir qilish maydoni va tugmacha. Komponentalarni loyihalash shakliga olib o‘tamiz hamda ilovani asta-sekin rivojlantira boshlaymiz. Tashib olib o‘tish metodi (drag-and-drop) quyidagilardan iborat: sichqoncha tugmachasini tanlab olingan komponenta ustida bosing, kursorni shaklning to‘g‘ri kelgan yeriga o‘tkazing, keyin esa sichqoncha tugmachasini yana bosing. Boshida faqat «standart» Palitra Komponentalari bilan cheklanamiz:

=> **Standart** qo‘shimcha ilovani tanlab oling.

=> Ro‘yxat komponentasini ListBox shakliga olib o‘ting.

=> Tahrir qilinayotgan kiritish maydoni EditBox ni olib o‘ting.

=> Button tugmachasi komponentasini olib o‘ting.

=>Komponentalarni o‘zingizning ilovangizdagi darchada qanday ko‘rmoqchi bo‘lsangiz, shunday joylashtiring va o‘lchamlarini shunday o‘zgartiring.

Obyekt noziri yordamida komponentalar xususiyatlarining boshlang‘ich qiymatlarini aniqlang. Items ro‘yxatining xususiyatlar qiymatlari katagida tugmachani bosing, ochilgan muharrir darchasiga she’rning dastlabki 7 satrini kiriting. Shakl va tugmachaning Caption xususiyatida ularning ma’noli nomlarini ko‘rsating (mos ravishda, «O‘nta negr bolasi» va «Natija»). Tahrir qilish maydonining Text xususiyatida natijani aytib berish satrini kiriting («To‘qqizta negr bolasi»).

Endi Kod Muharririga ulanish hamda, avval qabul qilinganidek, C++tilidagi har qanday dasturni yozish mumkin, shu jumladan, ANSI/ISO standartining so‘nggi kengaytmalarini ham. Biroq, avval ilovalarni tez ishlab chiqishning yangi vositalari hamda C++Builder da mavjud bo‘lgan qo‘shimcha komponentalar atributlaridan foydalanishga harakat qilib ko‘ramiz.

Xususiyatlar, metodlar va voqealar

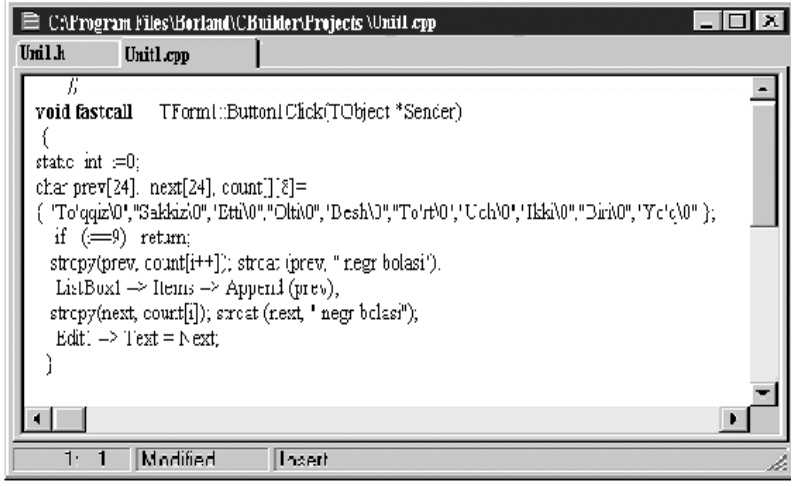
Ilovalarning tez ishlab chiqilishi obyektga mo‘ljallangan dasturlash doirasida xususiyatlar, metodlar va voqealarning qo‘llab-quvvatlanishini bildiradi. *Xususiyatlar* komponentalarning nomlar, matniy aytib berishlar yoki ma’lumotlar manbalari kabi turli xildagi tavsiflarini osongina o‘rnatish imkonini beradi. *Metodlar* (a’zo-funksiyalar) komponentadagi obyekt ustida ma’lum operatsiyalarni amalga oshiradi. Bunday operatsiyalar jumlasida qayta tiklash yoki multimedia qurilmasini qayta o‘rash kabi murakkab operatsiyalarni ham ko‘rsatish

mumkin. *Voqealar* komponentalarga foydalanuvchi ko‘rsatayotgan faollashtirish (aktivizatsiya), tugmalarni bosish yoki tahrir qilinadigan kiritish kabi ta’sirlarni ushbu ta’sirlarga sizning munosabat kodlaringiz bilan bog‘laydi. Bundan tashqari, voqealar komponentalar holatlarida sodir bo‘ladigan ayrim o‘ziga xos o‘zgarishlar paytida ham yuzaga kelishi mumkin. Bunday o‘ziga xos o‘zgarishlar qatorida ma’lumotlar bazasiga kirishning interfeysli elementlarida ma’lumotlarni yangilashni ko‘rsatib o‘tish kifoya. Xususiyatlar, metodlar va voqealar birgalikda ish olib borar ekan, ular Windows uchun ishonchli ilovalarni intuitiv tarzda dasturlash muhiti — RAD ni hosil qiladi.

=>Tanlangan obyekt bilan assotsiatsiyalanadigan (birgalikda yodga olinadigan) voqealarni ko‘rish uchun Obyektlar Nozirida Voqealar (Events) qo‘shimcha ilovasini ko‘rsating.

=>O‘zingiz shaklga joylashtirgan tugma komponentasini sichqoncha bilan ikki marta uring.

=>Ochilgan Kod Muharriri darchasida kursor ButtonIClick funksiyasi tanasiga instruksiyalarni kiritish uchun pozitsiyani ko‘rsatadi. Bu funksiya esa tugmachani bosishda yuzaga keladigan OnClick voqeasiga ishlov berish uchun mo‘ljallangan.



4.2-rasm. Kod Muharriri bajarilayotgan modul matnining Unit1.cpp faylida kiritilishi va tahrir qilinishini ta’minlaydi.

4.2-rasmda oddiy kod ko‘rsatilgan bo‘lib, u «Natija» tugmasini yana bir bor bosilishiga javoban avval turgan plev aytib berishini ro‘yxat oxiriga hamda navbatdagi next aytib berishini tahrir qilish maydoniga qo‘shadi. ListBoxI->Items->Append(prev) yo‘riqnomasi,

Append metodi yordamida, rrev satrini ListBoxI ro‘yxati obyektining Items xususiyatiga qo‘shadi. EditI->Text=next yo‘riqnomasi tahrir qilinayotgan EditI kiritish obyektining Text xususiyatiga next satrini taqdim etadi. Aytib berish satrlari ikki o‘lchamli count massivida saqlanadi va static turdagi butun o‘zgaruvchi tomonidan indekslanadi. Bu o‘zgaruvchi esa ButtonI tugmasini bosish bilan yuzaga keladigan voqeaga ishlov berish funksiyasining chaqirilishlari o‘rtasida o‘zining joriy qiymatini saqlaydi.

Birinchi versiyali ilovani loyihalash bosqichi shuning bilan tugallanadi va ishchi dasturni yaratishga kirishish mumkin bo‘ladi.

=>Run | Run bosh menusi buyrug‘i bilan ilovani kompilyatsiya qilish (ko‘chirish) va yig‘ish jarayonini ishga tushirib yuboring.

=>Dastur chaqirilgach, bir necha marta «Natija» tugmasini bosing.

Ikki yo‘nalishli ishlanma texnologiyasi

C++ Builder dasturchi va uning kodi o‘rtasiga hech qanday to‘siqlarni qo‘ymaydi. Two-Way Tools ikki yo‘nalishli ishlanma texnologiyasi vizual loyihalash instrumentlari va Kod Muharriri o‘rtasida moslashuvchan, integrallashgan va sinxronlashtirilgan o‘zaro aloqa vositasida sizning kodingiz ustidan nazoratni ta’minlaydi. Ikki yo‘nalishli ishlanma instrumentlari qanday amal qilishini kuzatib borish uchun quyidagi operatsiyalarni bajaring:

=>Sichqonchaning o‘ng tugmasini bosib, Kod Muharririning kontekstli menusini oching, keyin Swap Cpp/Hdr Files operatsiyasi yordamida Unit1.h. e’lonlar fayliga ulaning.

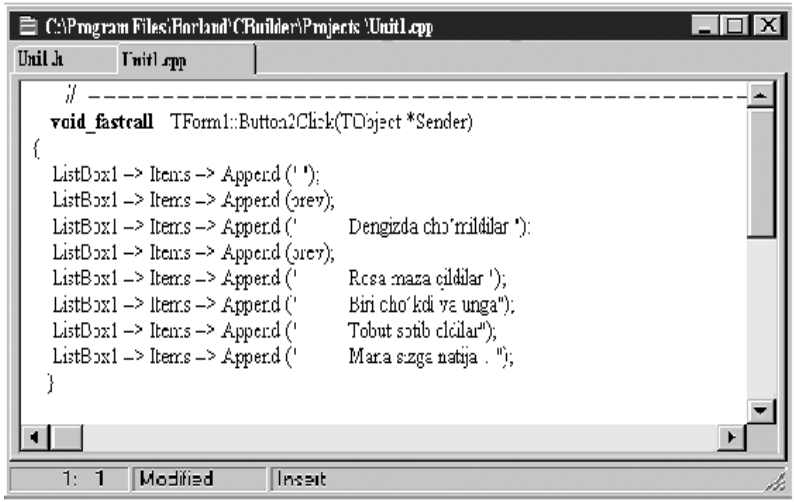
=> Instrumentlarning ekrandagi aksini shunday tashkil qilingki, bunda Kod Muharriri darchasida bir paytning o‘zida loyihalanayotgan shakl va Unit.h fayli ko‘rinsin.

=>OK Button tugmasining yana bitta komponentasini shaklga olib o‘ting. Tugmaning Caption xususiyatida uning ma’noli nomini «Yangi band» deb ko‘rsating.

Quyidagilarni kuzatib boring: siz tugmani shaklga olib o‘tishingiz bilanoq Unit1.h faylida Vutton 2 obyektining e’loni paydo bo‘lishi kerak, OnClick voqeaning aniqlanishi esa ushbu voqeaning qayta ishlovchisi bo‘lgan Button2Click metodining e’lon qilinishini generatsiyalaydi. Shaklni loyihalashning va kodni avtomatik generatsiyalash jarayonlarining mana shunday sinxronlashtirilishi C++ ilovaning vizual ishlanmasini haqiqatan ham tezlashtiradi va shuning bilan birga dasturning dastlabki matni ustidan nazoratni to‘la saqlab qoladi.

O‘zimizning birinchi ilovamizni ishlashda yana bir qadam qo‘yamiz — uni she’r bandini avtomatik tarzda generatsiyalashga majbur qilamiz. Buning uchun OnClick voqeasi ishlanmasining funksiyasini «Yangi band» tugmasini bosib, mazmun bilan to‘ldirishga to‘g‘ri keladi.

4.3-rasmda oddiy kod ko‘rsatilgan bo‘lib, u «Yangi band» tugmasining navbatdagi bosilishiga javoban yangi bandning ketmaket yettita satrini chiqarib beradi, bunda birinchi va uchinchi satrlar prev o‘zgaruvchisidan olinadi. Bu o‘zgaruvchi qiymatini «Natija» tugmasi voqeasining qayta ishlatgichi o‘zlashtirib olishi tufayli, bu qiymatni shakl sinfining foydalanuvchilar e’lonlarida public seksiyasida qayta aniqlashga to‘g‘ri keldi. Bu ish ikkala tugma voqealarining qayta ishlatgichlariga bu qiymatga kirish uchun imkon yaratish maqsadida qilindi.



4.3-rasm. Unit1.cpp faylida voqeaning yangi qayta ishlatgichi.

She’rni butunicha ko‘rib chiqish imkonini yaratish maqsadida ro‘yxat vertikal aylantirish chizig‘iga ega bo‘ldi.

C++ Builder har bir ilova bilan yashirin nomlari quyidagicha bo‘lgan uchta dastlabki faylni eslatishini yodda saqlab qolish kerak:

■ UnitI.cpp ilovangizning bajarilayotgan ishga tushirish kodini saqlaydi. Aynan shu yerda siz foydalanuvchining komponentalar obyektlariga ta’siri paytidagi dastur reaksiyasiga javob beradigan voqealarning qayta ishlatgichlarini yozib qo‘yasiz.

■ UnitI.h barcha obyektlar va ularning konstruktorlarining e’lonlariga ega. Voqealarni qayta ishlash funksiyalari e’lonlaridagi _fastcall kalit-so‘zga e’tibor bering (C++ Builder bu funksiyalarni

avtomatik tarzda generatsiya qiladi). _fastcall tufayli parametrlar stek orqali emas, balki markaziy protsessor registrlari orqali uzatiladi. Voqealarni qayta ishlatgichlarning chaqirishlari tez-tez ro‘y berib turadi, shuning uchun stek xotirasidan parametrlarni tanlab olishga sarflanadigan vaqtning tejalishi ancha sezilarli natijalarni beradi. C++ Builder kompilyatsiya qiladigan va to‘playdigan ilovalarning yuqori darajada tez harakatlanishining sabablaridan biri ham shu yerda yashiringan.

■ Project1.cpp ilovada mujassamlangan barcha obyektlarga xizmat ko‘rsatadi. Har qanday yangi shakl, dasturiy modul yoki ma’lumotlar moduli avtomatik tarzda loyihaviy faylga kiritiladi. Siz bosh menu buyrug‘i — View | Project Source yordamida yoki Loyiha Administratorining kontekstli menusidan shu nomdagi opsiyani tanlab olib, Kod Muharriri darchasida loyihaviy fayl dastlabki matnining mazmunini ko‘rib chiqishingiz mumkin. Hech qachon loyihaviy faylni qo‘lda tahrir qilmang!

Balki siz birinchi ilova ishlanmasini tugatib, dastlabki fayllarni keyingi seans uchun saqlab qolishni xohlarsiz. Buning uchun quyidagi xatti-harakatlardan birini bajarish kerak:

=>File | Save All buyrug‘i ilovaning hamma dastlabki fayllarini saqlaydi.

File | Save buyrug‘i dasturiy modulning ikkala buyrug‘ini saqlaydi, File | Save As buyrug‘i esa ularga yangi nom berishga ruxsat etadi.

File | Save Project As buyrug‘i, fayllarning joriy nomlaridan foydalanib, loyihaviy fayl tarkibiy qismlarining hammasidagi o‘zgarishlarni saqlaydi.

Loyihaviy shablonlarni qo‘llash

Obyektlar Xazinasidagi tayyor loyihaviy shablonlardan foydalanar ekansiz, siz dasturni ishlab chiqishda ko‘pchilik ilovalar uchun tipik bo‘lgan operatsiyalarni chetlab o‘tish imkoniyatiga ega bo‘lasiz. Bu qanday operatsiyalar dersiz. Bular, masalan, menu va tez chaqirib olish tugmalari panelini tuzish, standart chaqirishlar dialogi va fayllarni tuzishni tashkil etish bilan bog‘liq operatsiyalardir. Siz shablonga kiritgan o‘zgartishlar xuddi shu loyihaviy shablondan boshqa ishlab chiquvchilarning foydalanishiga ta’sir qilmaydi.

Ko‘p hujjatli interfeys (MDI) rejimida ishlash uchun loyihaviy shablon asosida ilova prototipini yaratishda quyidagi xatti-harakatlarni amalga oshiring:

Filtrlar Muharriri darchasida TOpenDialog komponentasining Filter xususiyati qiymatlari ustunida matniy hujjatlar fayllarining nomlari va kengayishlarini ko‘rsating.

Agar siz shunday ilovani kompilyatsiya qilib, to‘play olsangiz, bu holda u faqat MDI rejimida darchalar bilan amallar bajara olishini hamda darchani tanlab olingan fayllarning matniy mazmuni bilan to‘ldirmay turib fayllarni ochish dialogini chaqirib olishni «bilishi»ni ko‘rishingiz mumkin. Ya’ni prototip nofunksional va amaliy jihatdan befoyda bo‘lib qoldi. Ilova qandaydir ongli xulq-atvorga ega bo‘lishi uchun quyidagi xatti-harakatlarni bajaring:

Bosh menudan **View | Forms** buyrug‘ini bering va ro‘yxatdan MDIChild nomli sho‘ba shaklni tanlab oling.

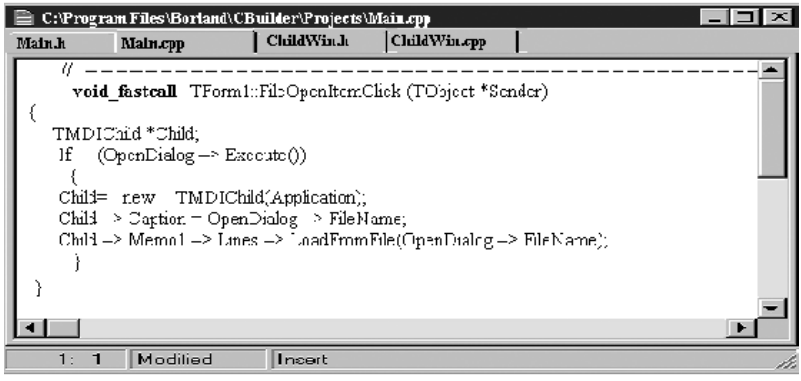
Memo tahrir qilish maydonining ko‘p satrli komponentasini Palitraning **Standart** qo‘shimcha ilovasidan sho‘ba shaklga olib o‘ting.

Lines xususiyatli satriy muharrirni tugmani bosish bilan chaqirib olib, TMemo komponentasining tahrir maydonini tozalang. Tahrir maydoni sho‘ba darchasining hammasini egallashi uchun Align xususiyatli alClient qiymatini o‘rnating. Uzun matniy fayllarni ko‘rib chiqishni osonlashtirish maqsadida ScrolBars xususiyatli Ssboth qiymatini o‘rnating.

=>Bosh shaklni sichqoncha yordamida faollashtirib, yana unga qayting hamda ilovalar menusidan **File|Open** buyrug‘ini tanlab oling.

=>Kod Muharriri darchasida kursor menuning tegishli elementini tanlashda yuzaga keladigan OnClick voqeasining qayta ishlatgichiga yo‘riqnomani kiritish uchun kerakli pozitsiyani ko‘rsatib beradi. C++Builder TOpenDialog bosh shakli (komponentalar Palitrasining Dialogs qistirmasidan) komponentasi uchun ushbu funksiyaning e’lonini avtomatik tarzda generatsiya qiladi.

4.4-rasmda shu voqeaning qayta ishlatgichi bo‘lgan FileOpenItemClick funksiyasi tanasini tashkil qiluvchi zarur yo‘riqnomalar ko‘rsatilgan.



4.4-rasm. Main.cpp faylida sho‘ba darcha yuklanishining amalga oshirilishi.

Ajratib olingan yo‘riqnoma Child sho‘ba darchasi Memo1 obyektining Lines satrlarini OpenDialog—>FileName nomli ochiq matniy faylning ichidagilari bilan yuklatadi.

Bu faylning ishlanishi hali tugallanganicha yo‘q, albatta. Siz uni kompilyatsiya qilib, to‘plab bo‘lsangiz, bir paytning o‘zida bir necha darchalardagi matniy fayllarni tahrir qila olasiz. Biroq natija beruvchi fayllarning saqlanishi hozircha ko‘zda tutilgan emas — o‘quvchining o‘zi File [Save va File | Save As menulari buyruqlari uchun osongina kod yozib oladi.

Ilovani mantiqan eng sodda matniy muharrirga aylantirish uchun bu Edit nomli bosh menu elementining tushib qoluvchi ro‘yxatiga qidirish va almashtirish buyruqlarini qo‘shish kerak.

4.2. Komponentalar palitrasi

C++ Builder 32 razryadli takomillashtirilgan Vizual Komponentalar Kutubxonasi VCL (Visual Component Library) bilan birgalikda yetkazib beriladi. Bu kutubxona eng murakkab ilovalarni qurish uchun mo‘ljallangan 100 dan ortiq takroran qo‘llanadigan komponentalardan iborat. Kutubxonaning asosiy komponentalari Palitralar komponentalarining instrumental Panelida berilgan. Komponentalar belgilari dasturingiz shakliga olib o‘tiladi.

Kutubxona Windows va Windows 95 operatsiya tizimlaridagi Foydalanuvchi Grafik Interfeysi standart interfeys obyektlarining to‘liq inkapsulatsiyalanishini o‘z ichiga oladi. Ular orasida, ixtisoslashgan komponentalar bilan bir qatorda, relyatsion ma’lumotlar bazasini boshqarish uchun mo‘ljallangan komponentalar alohida o‘rin egallaydi. Ishonchli va samarali dasturlarni yaratishda C++Builder obyektga mo‘ljallangan dasturlash (OMD) imkoniyatlaridan to‘liq foydalanadi. C++Builder bu OMD ekan, OLE (OCX) boshqaruvchi elementlarni kiritish uncha qiyinchilik tug‘dirmaydi. O‘z masalalaringiz talablarini kerakli darajada qondirish uchun Kutubxonaning mavjud komponentalaridan foydalaning, hosila komponentalar imkoniyatlarini kengaytiring.

C++Builder bosh xususiyati avvalambor uning dasturni vizual ishlash jarayonida nafaqat tayyor komponentalardan foydalanish, balki yangi komponentalarni yaratish qobiliyatida ham namoyon bo‘ladi. Yangi komponentalar, dastlabki komponentalar kabi, sodda bo‘lishi mumkin, bunda ularning funksional imkoniyatlari biroz kengaytirilgan yoki o‘zining mutlaqo o‘ziga xos ko‘rinishi, xulq-atvori va kodining mazmuni

bilan farqlanadigan bo‘ladi. Komponentalarning yaratilishi OMD ning vorislik mexanizmiga tayanadi, cheklanishlarga deyarli ega bo‘lmaydi hamda quyidagi bosqichlardan o‘tadi:

- mavjud komponenta turiga vorislik;
- yangi xususiyatlar, metodlar va voqealarni aniqlash;
- yaratilgan komponentani qayd etish.

Qidirish oson bo‘lishi uchun Palitra funksional jihatdan o‘xshash komponentalarni birlashtiradigan qo‘shimcha ilovalar bilan bo‘lingan. Tanlab olingan komponentaning kontekst menusini unga sichqonchaning o‘ng tugmasini bosib ochish mumkin.

Standart komponentalar

Komponentalar palitrasining Standart qo‘shimcha ilovalari komponentalari sizning dasturingizga Windows standart interfeysli elementlaridan 14 tasining ulanishini amalga oshiradi.

TMainlVlenu

TMainlVlenu bosh menu buyruqlari panelini va ularga mos keladigan tushib qoladigan menularni yaratadi. Barcha menu buyruqlarining identifikatorlari menuning har qanday konkret buyrug‘iga kirish huquqiga ega bo‘lgan ltems xususiyati bilan aniqlanadi, AutoMerge xususiyati Merge va Unmerge metodlari bilan birgalikda turli shakldagi menularning birlashish jarayonini boshqaradi.

TPopUpMenu

TPopUpMenu shakl yoki bironta boshqa komponenta uchun maxsus menu yaratadi. E’tiborga oling, aynan shu maqsad uchun har qanday boshqa komponenta PopUpMenu xususiyatiga ega bo‘lib, bu xususiyatda siz bilan bog‘liq menuga iqtibos qilishingiz mumkin.

Agar siz sichqonchaning o‘ng tugmasini shaklga yoki berilgan komponenta mansub bo‘lgan biron boshqa elementga bosish bilan maxsus menu ekranda paydo bo‘lishini xohlasangiz, AutoPopup xususiyatining true qiymatini o‘rnating. Voqea qayta ishlatgichi — OnPopup yordamida bevosita maxsus menuning paydo bo‘lishi oldidan bajariladigan protsedurani aniqlash mumkin.

TLabel

TLabel shaklda tahrir qilib bo‘lmaydigan statik matnning to‘rtburchak sohasini aks ettiradi. Odatda matn boshqa komponenta nomidan iborat bo‘ladi.

Nom matni Caption xususiyatining qiymatidir. Alignment xususiyati matnni tekislash usulini aniqlaydi. Shrift o‘lchami avtomatik

tarzda sohaning maksimal to‘ldirilishiga mos kelishi uchun AutoSize xususiyatining true qiymatini o‘rnating. Kalta soha ichida matnning hammasini ko‘rish imkoniga ega bo‘lish uchun WordWrap xususiyatining true qiymatini bering. Transparent xususiyatining true qiymatini o‘rnatsangiz, boshqa komponentaning bir qismini to‘g‘ri uning ustida joylashtirilgan nom orasidan ko‘rinib turadigan qilishingiz mumkin.

TEdit

TEdit axborot yakka satrining tahrir qilinayotgan kiritishidagi to‘rtburchak sohani shaklda aks ettiradi. Tahrir sohasining ichidagi boshlang‘ich narsalarni Text xususiyatining qiymati bo‘lgan satr aniqlaydi.

TEdit komponentasi TCustomEdit sinfining to‘g‘ridan-to‘g‘ri hosilasi bo‘lib, uning barcha xususiyatlari, metodlari va voqealariga vorislik qiladi.

TMemo

TMemo axborot ko‘plab satrining tahrir qilinayotgan kiritishidagi to‘rtburchak sohani shaklda aks ettiradi. Tahrir sohasining ichidagi boshlang‘ich narsalarni Lines xususiyatining qiymati bo‘lgan satrlar massivi aniqlaydi. Ushbu xususiyat qiymati ustunida tugmachani bossangiz, ro‘yxat elementlari muharririning darchasi ochiladi.

TMemo komponentasi TCustomMemo sinfining to‘g‘ridan-to‘g‘ri hosilasi bo‘lib, uning barcha xususiyatlari, metodlari va voqealariga vorislik qiladi.

TButton

TButton yozuvli to‘rtburchak tugmani yaratadi. Tugmacha bosilganda, dasturda biror bir xatti-harakat nomlanadi (initsiallashtiriladi).

Tugmachalar ko‘proq dialogli darchalarda qo‘llanadi. Default xususiyatining true qiymati tomonidan tanlab olingan yashirin tugmacha, dialog darchasida har gal Enter klavishasi bosilganda, OnClick voqea qayta ishlatgichini ishga tushiradi. Cancel xususiyatining true qiymati tanlab olgan uzish tugmachasi, dialog darchasida har gal Escape klavishasi bosilganda, OnClick voqea qayta ishlatgichini ishga tushiradi.

TVutton komponentasi TButtonControl sinfining hosilasi hisoblanadi.

TCheckBox

TCheckBox ikkita holatga hamda tavsifiy matnga ega bo‘lgan kvadrat chek-boksni yaratadi (bunda tavsifiy matn chek-boksning vazifasini spetsifikatsiya qiladi).

Boks holatini bildiruvchi «check» biror bir variantning tanlanishiga mos keladi (boks ustidan tortilgan chiziq bilan belgilanadi), «uncheck» holati esa tanlov olib tashlanishiga mos keladi — bunda Checked komponentasining xususiyati mos ravishda o‘zgaradi hamda OnSlick voqeasi yuzaga keladi. Tavsifiy matn Caption xususiyatida saqlanadi. AllowGrayed xususiyatining true qiymatini o‘rnatib, boksni to‘qroq rangli (masalan, kulrang) qilish mumkin. State xususiyati joriy holatni va boks rangini aks ettiradi.

TCheckBox komponentasi TButtonControl sinfining hosilasidir.

TRadioButton

TRadioButton ikkita holatga hamda tavsifiy matnga ega bo‘lgan yumaloq tugmachani yaratadi (bunda tavsifiy matn yumaloq tugmachaning vazifasini spetsifikatsiya qiladi).

Radio-tugmalar bir-birini istisno qiladigan tanlov variantlarining to‘plamidan iborat: ya’ni ushbu vaqt daqiqasida faqat bitta tugma tanlab olinishi mumkin (ichki qora doiracha bilan belgilanadi), avval tanlangan tugmadan esa tanlov avtomatik tarzda olinadi. Radio-tugma bosilganda, Checked komponentasining xususiyati ham mos ravishda o‘zgaradi va OnCkick voqeasi yuzaga keladi.

Odatda radio-tugmalar avvaldan shaklda o‘rnatilgan konteyner ichiga joylashtiriladi. Agar bitta tugma tanlangan bo‘lsa, ushbu guruhga mansub barcha boshqa tugmalarning tanlovlari avtomatik tarzda olib tashlanadi. Masalan, shakldagi ikkita radio-tugma, agar ular boshqa-boshqa konteynerlarda joylashgan bo‘lsagina, bir paytning o‘zida tanlab olinishi mumkin. Agar radio-tugmalarning guruhlanishi ochiq-oydin berilmagan bo‘lsa, bu holda ularning hammasi, yashirin holda, konteyner darchalari (TForm, TGroupBox yoki TPanel) dan birida guruhlanadi.

TRadioButton komponentasi TButtonControl sinfining hosilasidir.

TListBox

TListBox tanlash, qo‘shish yoki o‘chirish uchun mo‘ljallangan matn variantlari ro‘yxatining to‘rtburchak sohasini aks ettiradi.

Agar ro‘yxatdagi barcha elementlar ajratilgan sohaga sig‘masa, ro‘yxatni aylantirish lineykasi yordamida ko‘rib chiqish mumkin. Ro‘yxat elementlari ltems xususiyatining ichida, dastur bajarilish vaqtida tanlab olinadigan element raqami esa ltemIndex xususiyatining ichida joylashgan bo‘ladi. Ro‘yxat elementlari matn muharririning darchasi ltems xususiyati qiymatining grafasida tugmacha bilan ochiladi. Ro‘yxat elementlarini Items obyektining Add, Append, Delete

va Insert metodlari yordamida dinamik tarzda qo‘shish, o‘chirish, orasiga joylash va o‘rnini almashtirish mumkin. Masalan:

```
LisBox1->Items->Add("Ro‘yxatning oxirgi elementi");
```

Sorted xususiyatining true qiymati ro‘yxat elementlarini alifbo tartibida navlarga ajratib joylashtiradi.

TListBox komponentasi TCustomListBox sinfining hosilasi bo‘lib, uning barcha xususiyat, metod va voqealariga vorislik qiladi.

TComboBox

TComboBox tahrir sohasi hamda matn variantlarining tushib qoladigan ro‘yxati kombinatsiyasini tanlash uchun yaratadi.

Text xususiyatining qiymati bevosita tahrir sohasiga kiritib qo‘yiladi. Foydalanuvchi tanlab olishi mumkin bo‘lgan ro‘yxat elementlari Items xususiyatining dasturning bajarilish paytida tanlab olinishi mumkin bo‘lgan element raqami ItemIndex xususiyatining, tanlab olingan matnning o‘zi esa SelText xususiyatining ichida bo‘ladi. SelStart va SelLength xususiyatlari matnning qaysi qismini tanlab olishni belgilab berish yoki matnning qaysi qismi tanlab olinganini bilish imkonini beradi.

Items obyektining Add, Append, Delete va Insert metodlari yordamida ro‘yxat elementlarini dinamik tarzda qo‘shish, o‘chirish, orasiga qo‘yish va o‘rnini almashtirish mumkin, masalan:

```
ComboBoxl->Items->Insert(0, "Ro‘yxatdagi birinchi element");
```

Sorted xususiyatining true elementi ro‘yxat elementlarini alifbo tartibida navlarga ajratilishini ta’minlaydi. TComboBox komponentasining turini Style xususiyatidan tanlab olish mumkin.

TComboBox komponentasi TCustomComboBox sinfining hosilasi bo‘lib, uning barcha xususiyatlari, metodlari va voqealariga vorislik qiladi.

TScrollBar

TScrollBar darcha, shakl yoki boshqa komponenta ichidagilarini ko‘rib chiqish uchun, masalan, biror bir parametr qiymatini berilgan interval ichida harakatlanishi uchun yugurgichli aylantirish lineykasini yaratadi.

Aylantirilayotgan obyekt xulq-atvorini OnScroll voqealar qayta ishlatgichi aniqlaydi. Foydalanuvchi lineykaning o‘zida sichqonchani bosganda (yugurgichning har ikkala tomonida), yugurgich qanchaga surilishi kerakligini LargeChange xususiyatining qiymati aniqlab

beradi.Foydalanuvchi sichqonchani strelkali tugmachalar (lineyka oxiridagi) ustida bosganda yoki pozitsiyalash tugmachalarini bosganda, yugurgich qanchaga surilishi kerakligini SmallChange xususiyatining qiymati aniqlab beradi.

Min va Max xususiyatlarining qiymatlari yugurgichning yo'l qo'yilishi mumkin bo'lgan joy almashinuvlari intervallarini belgilaydi. Sizning dasturingiz yugurgichni Position xususiyatining qiymati aniqlab beradigan kerakli pozitsiyaga joylashtirishi mumkin. SetPcirums metodi bir paytning o'zida Min, Max va Position ga tegishli barcha xususiyatlar qiymatlarini aniqlab beradi.

TGroupBox

TGroupBox to'g'ri burchakli ramka ko'rinishidagi konteyner bo'lib, u qandaydir bir interfeys elementlarining mantiqan bog'langan guruhini shaklda vizual birlashtiradi. Bu komponenta Windowsning bir nomdagi obyektning inkapsulatsiyalanishidan iborat.

TRadioGroup

TRadioGroup to'g'ri burchakli ramka ko'rinishidagi konteyner bo'lib, u bir-birini mantiqan istisno qiladigan radio-tugmalar guruhini shaklda vizual birlashtiradi.

Radio-tugmalar bitta konteynerga joylashtirilganda «guruhlanadi». Bu guruhdan faqat bitta tugmacha tanlab olinishi mumkin. TRadioGroup komponentasiga tugmalarni qo'shish uchun Items xususiyatining tahriri bajarilishi kerak. Items xususiyatining navbatdagi satriga nom berilsa, shu tugma guruhlovchi ramkada paydo bo'ladi. Ushbu daqiqada qaysi tugma tanlab olinishi kerakligini ItemIndex xususiyatining qiymati aniqlab beradi. Columns xususiyatining tegishli qiymatini joylashtirib, siz radiotugmalarni bir necha ustunga guruhlashingiz mumkin.

TPanel

TPanel boshqa komponentalarni o'z ichiga olishi mumkin bo'lgan bo'sh panelni yaratadi. Siz TPanel dan o'z shaklingizda instrumentlar paneli yoki holatlar satrlarini yaratish uchun foydalanishingiz mumkin.

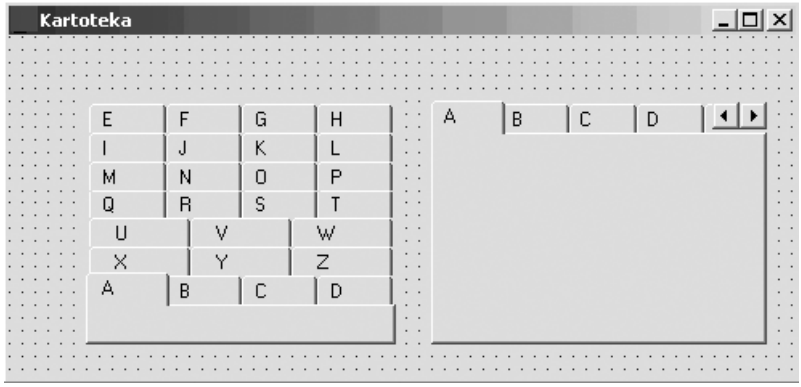
TPanel panel komponentasi TCustomPanel sinfining hosilasi bo'lib, uning barcha xususiyatlari, metodlari va voqealari to'liq vorislik qiladi.

Windows komponentalari

Windows komponentalari sizning dasturingizga Windows ning 12 ta interfeys elementlarining ulanishini amalga oshiradi.

TTabControl

TTabControl bir-birini qisman yopib turadigan qo‘shimcha kartoteka elementlarining to‘plamini aks ettiradi. Qo‘shimcha ilovalarning nomlari Tabs xususiyatining ro‘yxatiga ushbu xususiyat qiymati ustunidagi tugmacha bilan kiritiladi. 4.5-rasmda alifbo kutubxona ko‘rsatkichi bilan ishlash uchun ilova shaklining qolipi ko‘rsatilgan. Agar maydonlarning hammasi shaklda bir qatorga sig‘masa, MultiLine xususiyatining true qiymatini o‘rnatish yoki qo‘shimcha ilovalarni strelkali tugmachalar yordamida aylantirib turish mumkin.



4.5-rasm. Qo‘shimcha kartoteka ilovalari nomlari bilan.

Enabled xususiyatining false qiymati o‘rnatilsa, bu ba’zi bir qo‘shimcha ilovalarning tanlanishini taqiqlaydi.

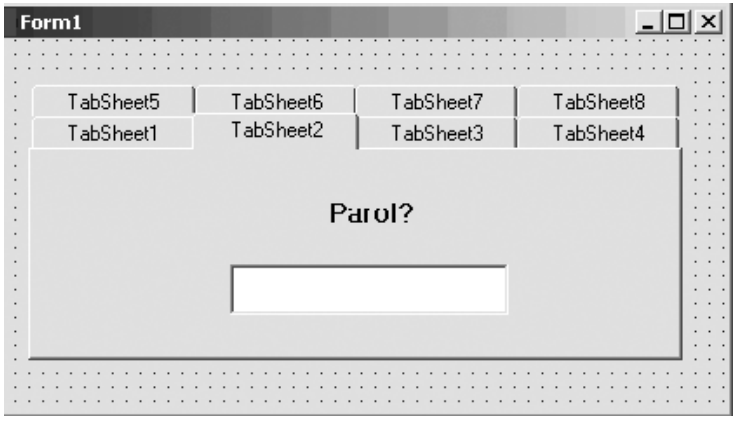
TPageControl

TPageControl ko‘p varaqli dialogni tashkil etish uchun mo‘ljallangan qisman bir-birining ustini yopuvchi qo‘shimcha kartoteka ilovalari ko‘rinishidagi maydonlar to‘plamini aks ettiradi.

Tegishli qo‘shimcha ilovaga ega bo‘lgan yangi dialogli sahifani yaratish uchun berilgan komponentaning kontekst menusidan New Page opsiyasini tanlab oling. Siz konkret sahifani quyidagi usullarning biri yordamida faollashtirishingiz mumkin: ActivPage xususiyatining tushib qolayotgan ro‘yxatidan tanlab olingan sichqoncha yordamida; shuningdek, kontekst menuning NextPage va Previous Page opsiyalari yordamida qo‘shimcha ilovalarni varaqlash vositasida. PageIndex xususiyati faol sahifa raqamiga ega. Tab Visible xususiyatining false qiymatini o‘rnatib, shu sahifani ko‘rinmas qilish mumkin.

4.6-rasmda ikkinchi faollashtirilgan sahifa uchun ko‘p sahifali dialogli ilova shaklining qolipi keltirilgan. Qo‘shimcha ilovalar bilan ishni

TTabSheet qurilma boshqarish komponentasi amalga oshiradi. Agar hamma qo‘shimcha ilovalar bir qatorga sig‘masa, komponenta aylantirish tugmalarini chiqaradi. Qo‘shimcha ilovalarni bir necha qatorda aks ettirish uchun MultiLine xususiyatining true qiymatini bering.

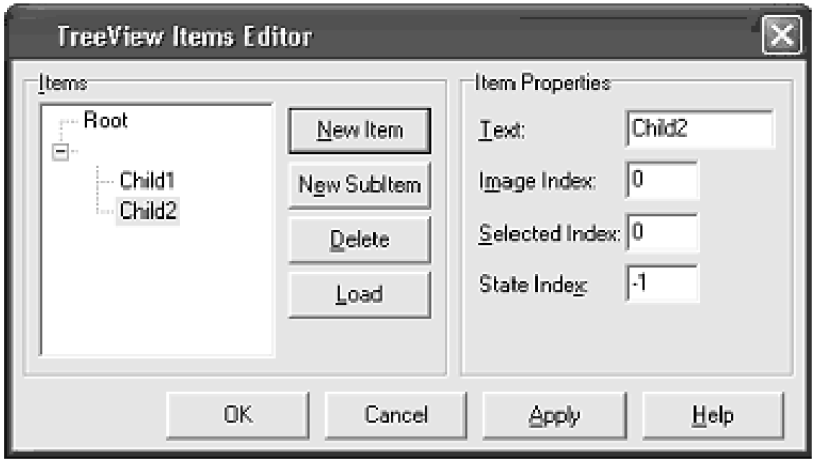


4.6-rasm. Ko‘p sahifali dialog.

TTreeView

TTreeView elementlar tartibi tabaqaviy (shajaraviy) ko‘rinishga ega bo‘lgan maydonni aks ettiradi: hujjatlar sarlavhasi, ko‘rsatkichdagi yozuvlar, disklardagi fayllar yoki kataloglar. Bu komponentaning ishini ko‘plab Windows ilovalarida ko‘rish mumkin.

Items xususiyati shajara elementlarining tahrir qilinayotgan ro‘yxatiga ega bo‘lgan TTreeNodes obyektiga iqtibos (murojaat) qiladi.



4.7-rasm. TTreeView komponentasi shajarasini tuzish.

Shajara elementlari muharririning darchasi (4.7-rasm) ushbu xususiyat qiymatlari ustunidagi tugmacha bilan ochiladi. Har bir shajara elementi belgidan, bu belgini eslatuvchi subelementlar ro‘yxatidan hamda bitli obrazlar qatori (agar shundaylar bo‘lsa) dan iborat. Sichqonchani element ustida shiqillatar ekan, foydalanuvchi tegishli subelementlar ro‘yxatini ochishi yoki yopishi mumkin. Sichqoncha ikki marta shiqillatilsa, shajara ajdodi tugunining bitta darajasi ochiladi hamda uning faqat to‘g‘ridan-to‘g‘ri vorislarini ko‘rsatadi. ShowButtons xususiyati ajdod tugunidan chap tomonda turgan tugmaning aks ettirilishiga (agar ushbu uzel ochilmagan bo‘lsa va subelementlarga ega bo‘lsa «+» belgili tugma, aks holda esa «—» belgili tugma) javob beradi: bu tugmaning bosilishi ajdod element ustida sichqonchaning ikki marta shiqillagani bilan teng.

Indent qiymati ochilayotgan avlodlar sonini belgilaydi. Avlodlar ro‘yxatini alifbo tartibida joylashtirishda SortType xususiyati uchun stText qiymatini o‘rnating. Visible xususiyatining true qiymati ajdodlar va avlodlarni bog‘laydigan shajara shoxlari — chiziqlarining aksini keltirib chiqaradi.

Elementlarni ro‘yxatga dinamik tarzda Items->TTreeNode Obyekti uchun quyidagi metodlar yordamida qo‘shish va kiritish mumkin: AddChildFirst, AddChild, AddChildObjectFirst, AddChildObject, AddFirst, Add, AddObjectFirst, AddObject.Insert, InsertObject.

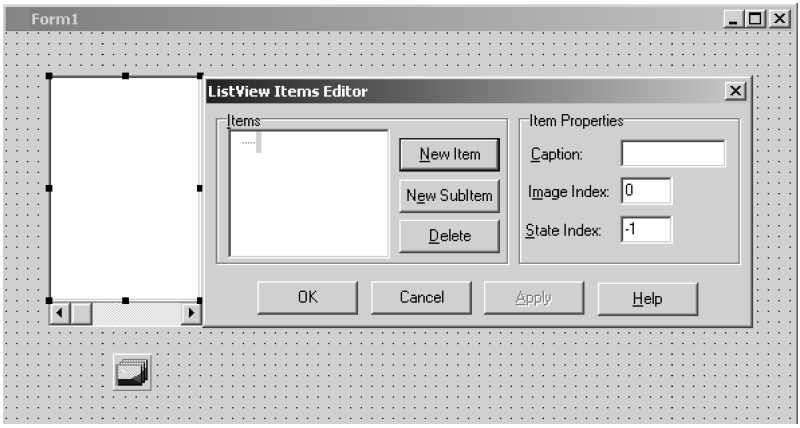
TListView

TListView tabaqaviy (shajaraviy) elementlar ro‘yxatiga ega bo‘lgan maydonni aks ettiradi. ViewStyle xususiyati ro‘yxat elementlarini quyidagi tartibda aks etishini belgilaydi: sarlavhali ustunlar bo‘yicha, vertikal, gorizontal, katta yoki kichik piktogrammalar bilan.

Items xususiyati ost yozuvlarini qo‘shish, o‘chirish va modifikatsiyalash, shuningdek, ro‘yxat elementlariga piktogrammalarni tanlash imkonini beradi. Ro‘yxat muharriri ushbu xususiyat qiymatlari grafasidan tugmacha yordamida chaqiriladi.

Columns xususiyati ro‘yxatdagi ustunlar sarlavhalari nomlarining tahrir qilinadigan ro‘yxatiga ega bo‘ladi. Ustunlar muharririning darchasi ushbu xususiyat qiymatlari grafasidan tugmacha yordamida chaqiriladi. Sarlavhalarni ko‘rish uchun ViewStyle xususiyatining vsReport qiymatini, ShowColumnHeaders xususiyatining true qiymatini topshiring. ColumnClick xususiyati true qiymatining o‘rnatilishi tugmaga teng keladigan sarlavha xulq-atvorini aniqlaydi: foydalanuvchi sichqonchani ost yozuv ustida shiqillatsa, OnColumnClick voqeasi yuzaga keladi. OnEdiling va OnEcUtecl voqealari foydalanuvchi ost yozuv tahririni boshlayotgan va tugallayotgan vaqtda yuzaga keladi.

Tushib qolayotgan xususiyatlar ro‘yxati LargeImages (SmallImages) dan piktogrammalar manbaini tanlash uchun ViewStyle xususiyatining vslcon(vsSmallIcon) qiymatlarini bering. IconOptions xususiyatining AutoArrang rejimida Arrangement xususiyatining tanlab olingan qiymatiga muvofiq piktogrammalar tekislanadi, WrapText xususiyati esa ost yozuv matnini, agar u piktogrammaga eni bo‘yicha sig‘masa, ko‘chirish kerakligini bildiradi. 4.8-rasmda shajarasimon ro‘yxat tuzish jarayoni ko‘rsatilgan, bunda TImageList komponentasi katta piktogrammalar manbai bo‘lib, LargeImages xususiyati ushbu komponenta obyektini ko‘rsatadi.



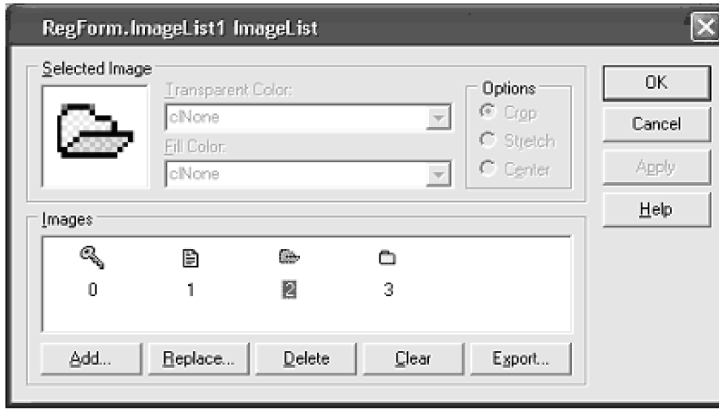
4.8-rasm. Shajarasimon ro‘yxatning tuzilishi va uning shaklda aks ettirilishi.

TImageList

TImageList bir xil K*k o‘lchamdagi grafik tasvirlardan iborat kolleksiya uchun konteyner yaratadi. Bu tasvirlarning har bittasini ularning 0 dan n-1 gacha qiymatlar intervalidagi indeks bo‘yicha tanlab olish mumkin. Grafik kolleksiyalar bitli obrazlar yoki piktogrammalarning katta to‘plamlariga samarali xizmat ko‘rsatish uchun qo‘llanadi. Ushbu bitli obrazlar va piktogrammalar eni k*n bo‘lgan yagona bitli obraz sifatida saqlanadi. Niqobli monoxrom tasvirlar tiniq trafaretlar sifatida aks ettiriladi. TImageList komponentasi saqlanayotgan tasvirlarni yozish, tanlash va chiqarish jarayonlarini osonlashtiruvchi metodlarga ega.

Tasvirlar kolleksiyasi muharririning darchasi komponentaga sichqoncha bilan ikki marta shiqillatish orqali yoki uning kontekst menusidagi ImegeListEditor opsiyasi bilan ochiladi (4.9-rasm).

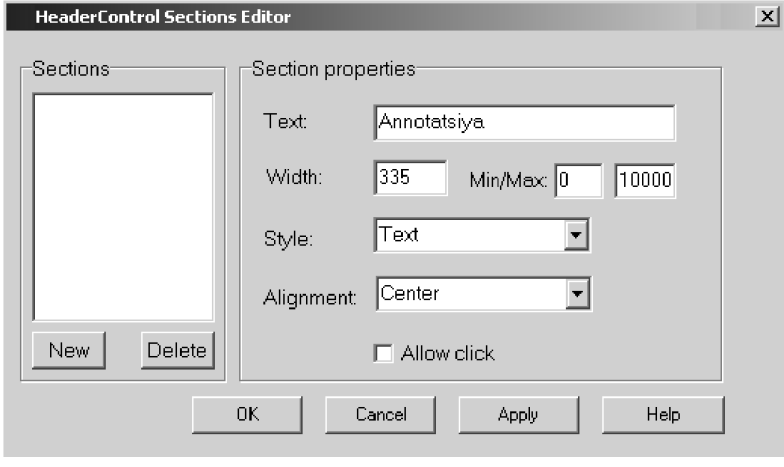
TImageList komponentasi TCustomImageList sinfining hosilasi bo‘lib, uning xususiyatlari, metodlari va voqealariga to‘liq vorislik qiladi.



4.9-rasm. Piktogrammalar kolleksiyasini tuzish.

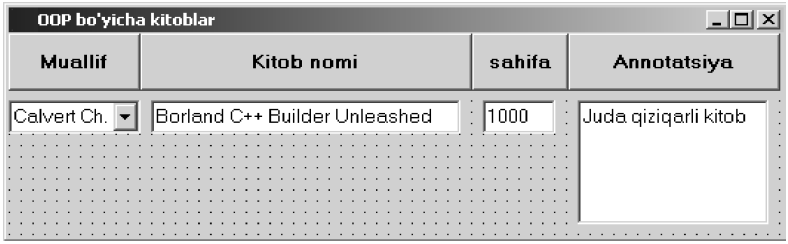
THeaderControl

THeaderControl dasturning bajarilish jarayonida enini o‘zgartirish mumkin bo‘lgan ustunlar sarlavhalarining to‘plami uchun konteyner yaratadi. Sections xususiyatida sanab o‘tilgan sarlavhalarni axborot maydonlari ustida, masalan, TListBox komponentalari ro‘yxatlari ustida joylashtirish mumkin. Sarlavhali seksiyalar muharririning darchasi (4.10-rasm) ushbu xususiyat qiymatlarining grafasida tugmacha bilan ochiladi.



4.10-rasm. Sarlavhalar seksiyasini tuzish.

Sarlavhalar seksiyasini har qanday boshqa komponentalarga moslashtirilgani uchun (4.11-rasm) ularning eni bilan ishlash ularga yaqin komponentalar enining adekvat ravishda o‘zgarishiga olib kelmaydi.

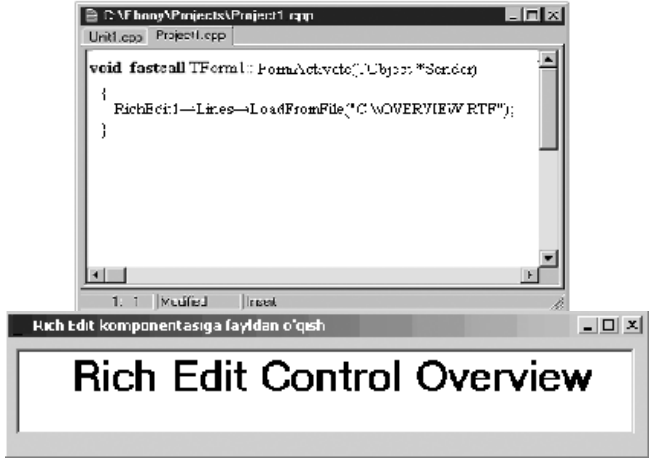


4.11-rasm. Bibliografik ko‘rsatkichni tuzish.

Agar siz ustun eni seksiya enining o‘zgarishiga muvofiq o‘zgarishini xohlasangiz, mana shu xatti-harakatlar uchun javob beradigan OnSectionResize voqealar qayta ishlatgichini yozishingizga to‘g‘ri keladi.

TRichEdit

TRichEdit RTF (Rich Tect Format) formatidagi ko‘p satrli axborotning tahrir qilinadigan kiritish sohasini aks ettiradi. ATF formati shrift atributlari va paragraflarni formatlashning turli variantlarini o‘z ichiga oladi. Bu formatni ko‘plab professional matniy protsessorlar, masalan, MicrosoftWord qabul qiladi.



4.12-rasm. RTF faylini o‘qish uchun ilovalarni loyihalash va bajarish.

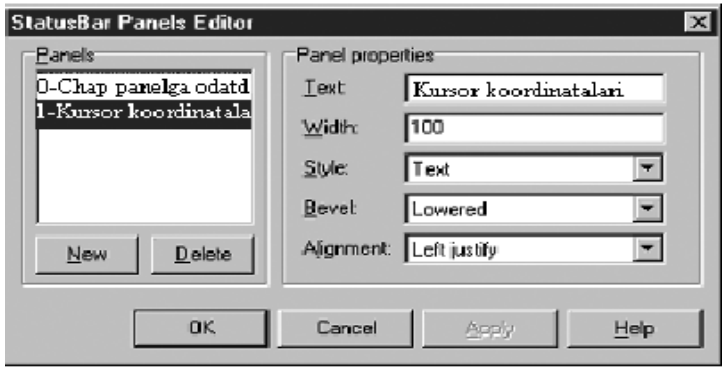
Kod Muharriri darchasida (4.12-rasmning markaziy qismi) OnActivate voqeasi qayta ishlovchisining yagona instruksiyasi ajratib ko‘rsatilgan bo‘lib, u shakl faollashganda yuzaga keladi va OVER VIEW.RTF faylining o‘qilishini tahrir qilinayotgan kiritish komponentasining RichEditl obyektiga chaqirib oladi. Rasmning pastki qismida kompilyatsiya qilingan va to‘plangan ilovaning ishi ko‘rsatilgan.

TRichEdit komponentasi TCustomRichEdit sinfining to‘g‘ridan-to‘g‘ri hosilasi bo‘lib, uning xususiyatlari, metodlari va voqealariga to‘liq vorislik qiladi.

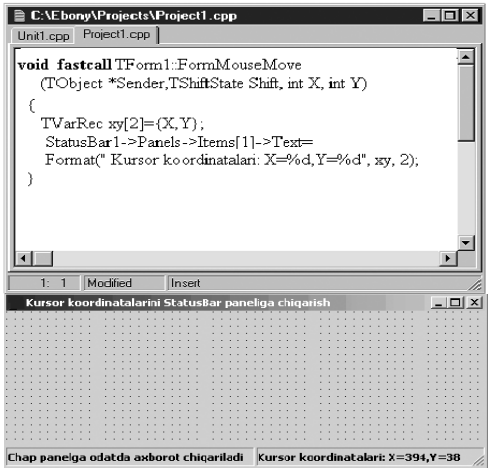
TStatusBar

TStatusBar dastur ishlayotgan paytda chiqarib beriladigan maqomli (statusli) axborotni aks ettirish uchun holatlar panellari satrini (bu satr odatda shaklning pastki chegarasi bo‘ylab tekislanadi) yaratadi.

Har bir panel Panels xususiyatlari ro‘yxatida o‘z o‘rniga ega. Panellar 0 indeksidan boshlab, chapdan o‘ngga qarab shakllantiriladi. Panellar Muharririning darchasi (4.13-rasm) ushbu xususiyat qiymatlarining grafasida tugmacha bilan ochiladi. SimplePanel xususiyatidan satr holatining aks etish turini (bir yoki ko‘p panelli) qayta ulash uchun foydalaniladi.



4.13-rasm. Holatlar panellari satrini tuzish.



4.14-rasm. Axborotning holatlar satri paneliga chiqarilishi.

Kod Muharririning darchasida (4.14-rasmning markaziy qismi) OnMouse Move voqeasining qayta ishlatgichiga tegishli yagona instruksiya ajratib ko‘rsatilgan. Bu voqea sichqonchaning shakl bo‘ylab harakatlanishi paytida yuzaga keladi hamda kursor koordinatorini holatlar satri komponentasining StatusBarl obyekti Panels->Item[I] paneliga chiqa-

radi. Rasmning pastki qismida kompilyatsiya qilingan va yig‘ilgan ilovaning ishlashi ko‘rsatilgan.

Bu komponenta bitta nomdagi Windows obyektining inkapsulatsiyalanishining o‘zginasidir.

TTrackBar

TTrackBar belgilar va joriy holat regulyatori (rostlagichi) ga ega bo‘lgan shkalani (aylantirish lineykasining varianti) yuzaga keltiradi.

Min va Max xususiyatlari shkala qiymatlari intervalini o‘rnatadi, bunda Position xususiyati regulyatorning berilgan interval ichidagi joriy pozitsiyasini aks ettiradi. Tasvirlanadigan belgilar soni Frequency xususiyatini spetsifikatsiyalaydi. Foydalanuvchi shkalaning o‘zida (regulyatorning ikki tomonida) sichqonchani shiqillatganda yoki **PageUp** va **PageDown** klavishalarini bosganda, regulyator nechta belgiga surilishi kerakligini PageSize xususiyatining qiymati aniqlaydi. Foydalanuvchi kursorni pozitsiyalash klavishasini bosganda, regulyator nechta belgiga surilishi kerakligini LineSize xususiyatining ko‘rsatkichi aniqlaydi.

Shkala turishini o‘zgartirish uchun TickStyle va TickMarks xususiyatlaridan foydalaning. SelStart va SelEnd xususiyatlarining qiymatlari regulyatorning ruxsat etilgan ko‘chishlari chegaralarini o‘rnatadi.

TProgressBar

TProgressBar sizning dasturingizdagi qandaydir protseduraning bajarilish jarayonini kuzatib boradi. Protsedura bajarilgan sayin, to‘g‘ri burchakli indikator chapdan o‘ngga qarab asta-sekin berilgan rangga bo‘yalib boradi.

Min va Max xususiyatlari indikator qiymatlari indikatorini o‘rnatadi. Step xususiyati indikator pozitsiyasi o‘zgarishi bilan har gal Position xususiyati qiymatining o‘zgarish qadamini belgilaydi.

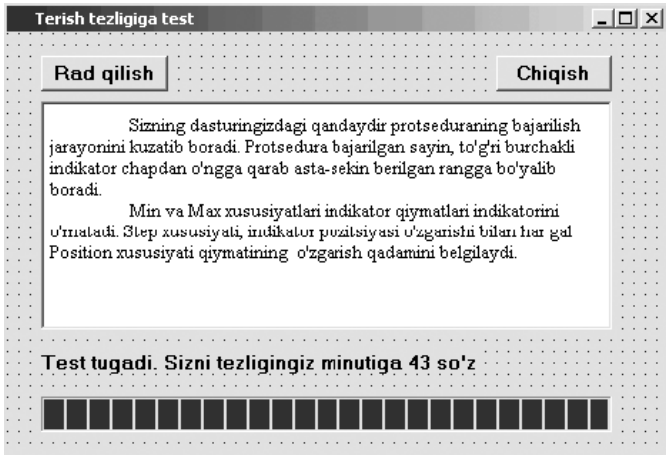
C++Builder hazilnamo misol bilan birga yetkazib beriladi: bu misol mashinistkalarning ish tezligini o‘lchashga qaratilgan testdagi progress-indikator ishini namoyish etadi (4.15-rasm).

=>Bosh menuning File | Open Project buyrug‘i bo‘yicha loyihalarni tanlash dialogini oching. => \...\CVuilder\ Examples\Apps\Wpm katalogiga kiring.=> Wpm loyiha faylini tanlab oling va Open tugmasini bosing.

=>Bosh menuning Run | Run buyrug‘i bo‘yicha ilovani kompilyatsiya qilish va yig‘ish jarayonini ishga tushiring.

Dasturiy modulning WPMMAIN.CPP kodi nihoyatda qisqa bo‘lib, qo‘shimcha izohlarga muhtoj emas. Siz ilovaning xulq-atvorini o‘zingizning xohishingizga ko‘ra osongina moslashtirishingiz mumkin,

masalan, uni o‘z ona tilingizga o‘tkazib olishingiz mumkin. Test sinovlarining natijalari shuni ko‘rsatadiki, u mashinada yozish uchun hech ham to‘g‘ri kelmaydi.



4.15-rasm. Mashinistkalarni test sinovidan o‘tkazish uchun ilovaning ishi.

TUpDown

TUpDown yuqoriga va pastga ko‘rsatuvchi strelkalarga ega bo‘lgan juftlangan tugmalarni yaratadi. Bu tugmalarning bosilishi mos ravishda Position xususiyatining qiymatini son jihatdan ko‘paytiradi yoki kamaytiradi.

Bu komponenta odatda Associate xususiyati tomonidan beriladigan kuzatib boruvchi boshqaruv elementi bilan birgalikda yetkazib beriladi. Agar tahrir qilinadigan kiritish sohasi kuzatib boruvchi element sifatida xizmat qilsa, Position xususiyatining qiymati ko‘rinadigan matnning formatlanishini aniqlaydi. Agar Associate xususiyati spetsifikatsiyalanmagan bo‘lsa, Position xususiyatining qiymati raqamli kattalikka ega bo‘ladi.

THotKey

THotKey dasturning bajarilish paytida tez chaqirish (shortcut) klavishalarini o‘rnatish uchun qo‘llanadi. Foydalanuvchi odatda modifikator (Ctrl, Alt yoki Shift) dan hamda har qanday belgi, shu jumladan, F1,...F12 funksional klavishalaridan iborat bo‘lgan «kuydiradigan» klavishalar kombinatsiyasini joriy qilishi mumkin.

Joriy qilingan hamda HotKey xususiyatiga yozib qo‘yilgan kombinatsiyani boshqa komponentaning ShortCut xususiyatiga berish

mumkin. Loyihalash bosqichida kuydiradigan klavishalarni tanlab olish uchun HotKey va Modifiers xususiyatlaridan foydalaning, ularni rad etish uchun esa InvalidKeys xususiyatidan foydalaning. Dastur bajarilayotgan paytda kombinatsiyani o‘zgartirish uchun modifikator klavishasini bosilgan holatda ushlab turib, bir paytning o‘zida yangi belgini kiriting.

Qo‘shimcha komponentalar

Komponentalar palitrasining **Additional** qo‘shimcha ilovalar komponentalari sizning dasturingizga Borland korporatsiyasi maxsus C++Builder muhiti uchun ishlab chiqqan 9 ta boshqarish elementini kiritadi.

TBitBtn

TBitBtn bit obrazining tasviri tushirilgan tugmachani yaratadi. Bunday tugmachalar ko‘proq maxsus dialogli darchalarda qo‘llanadi.

Grafik tugmachalar bit obrazlari, ularning ko‘rinishi va tugmachada joylashishini spetsifikatsiyalash xususiyatlariga ega bo‘ladi. Siz C++Builder qurilmasiga kirgan alohida tasvirlar katalogidagi grafik tugmalarning tayyor stillaridan foydalanishingiz ham yoki bo‘lmasa tasvirlarni tahrir qilish tizimlarining biri tomonidan yaratilgan suratlardan foydalanishingiz ham mumkin.

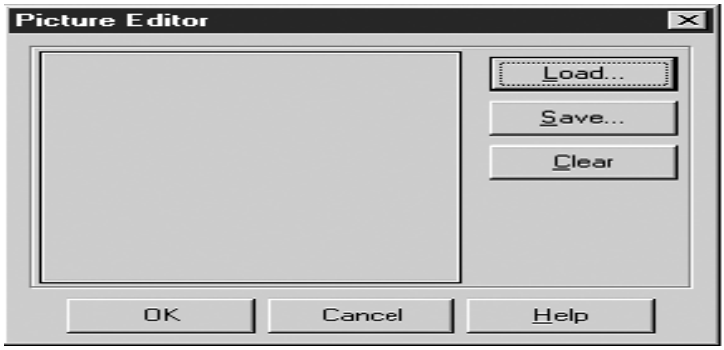
Tugmaning turli holatlariga (masalan, «bosilgan», «qo‘yib yuborilgan», «taqiqlangan» va h.k.) turli bit obrazlari mos kelishi mumkin.

Tasvirlar Fayllari Muharririning bmp kengayishli darchasi (4.16-rasm) Glyph xususiyatining qiymatlari tugmasi bilan ochiladi. King xususiyati sizga yozuvlar va tegishli grafika (OK, Cancel, Helo va boshqalar) bilan ta’minlangan standartlashtirilgan tugmalarni yaratish imkonini beradi.

TSpeedButton

TSpeedButton odatda ma’lum menu buyruqlarini tez chaqirish yoki rejimlarni o‘rnatish paneli (TPanel) da joylashtiriladigan grafik tugmani yaratadi.

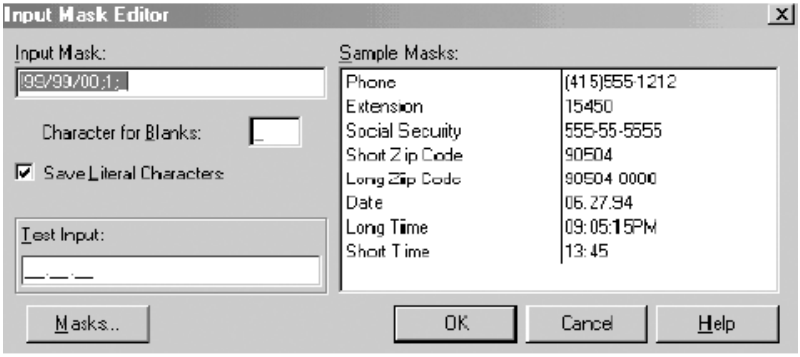
Tezkor tugmaning turli holatlariga (masalan, «bosilgan», «qo‘yib yuborilgan», «taqiqlangan» va h.k.) turli bit obrazlari mos kelishi mumkin. Bir-birining o‘rnini bosadigan tasvirlar va yozuv matnini tanlash uchun xususiyatlar mavjud. Kengayishli tasvirlar fayllari muharririning darchasi (4.16-rasm) Glyph xususiyati qiymatlarining grafasidagi tugma bilan ochiladi. Tez tugmalarning boshqa xususiyatlari ularning biror bir guruhdagi ishini tashkil etadi.



4.16-rasm. bmp kengayishli bit obrazlari fayllari tasvirlarining muharriri.

TMaskEdit

TMaskEdit o‘ziga xos formatdagi ma’lumotlarning tahrir qilinadigan nazoratdagi to‘rtburchak sohasini yaratadi. Kiritilayotgan matnning to‘g‘riligi ruxsat etilgan formatlarni kodlovchi niqob vositasida tekshiriladi. Bu formatlarga matn kiritilgan va foydalanuvchiga taqdim etilgan bo‘lishi mumkin (sana, vaqt, telefon raqami va h.k.). EditMask xususiyati joriy niqob kodini saqlaydi. Niqoblar muharriri darchasi (4.17-rasm) ushbu xususiyat qiymatlari grafasida tugma bilan ochiladi.



4.17-rasm. Telefon raqamlarini kiritish uchun niqobni yaratish.

TMaskEdit komponentasi TCustomMaskEdit sinfining to‘g‘ridan-to‘g‘ri hosilasidir. U satrlar yoki ustunlar bo‘yicha belgili ketma-ketliklarni aks ettirish uchun mo‘ljallangan muntazam (regulyar) to‘rni yaratadi.

TStringGrid

TStringGrid komponentaga tegishli barcha xususiyatlarning nomlari va vazifalari bo‘lib, siz ulardan dasturni loyihalash bosqichida to‘la foydalanishingiz mumkin. Ular keyingi paragrafda tavsifi berilgan TDrawGrid komponentasi xususiyatlariga to‘liq to‘g‘ri keladi.

Simvolli ketma-ketliklar bilan bog‘liq barcha obyektlar kerakli obyektga murojaat qilish imkonini beradigan Objects xususiyatida mujassam bo‘lgan. Dastur bajarilish paytida simvolli ketma-ketliklar va setka ustunining ular bilan bog‘liq obyektlari Cols xususiyati bilan adreslanadi. Rows xususiyati setka satrlari bilan xuddi shunday ish tutish imkonini beradi. Setkaning barcha simvolli ketma-ketliklari setkaning kerakli uyasini adreslaydigan (manzillaydigan) Cells xususiyatida mujassamdir.

TDrawGrid

TDrawGrid tuzilma holiga keltirilgan grafik ma’lumotlarni satrlar yoki ustunlar bo‘yicha aks ettirish uchun muntazam setka yaratadi. RowCount va ColCount xususiyatlari vertikal va gorizontal bo‘yicha setka uyalarining sonini belgilaydi.

Options xususiyatining qiymatlari setkaning turi (masalan, ustunlar orasida ajratuvchi chiziqlarga ega bo‘lgan setka turi) va uning xulq-atvorini (masalan, ustundan ustunga Tab klavishasi bo‘ylab o‘tish) o‘zgartirish imkonini beradi. Setkadagi ajratish chiziqlarining eng GridLineWidth xususiyati tomonidan belgilanadi, aylantirish chiziqchalari esa ScrollBars xususiyati tomonidan qo‘shiladi. FixedCold va FixedRows xususiyatlari ustunlar va satrlarning aylantirilishini taqiqlab qo‘yish imkonini beradi, FixedColor xususiyati esa barcha ustun va satrlarga ma’lum rang beradi.

DefaultDrawing xususiyatining true qiymati setka uyalarining ichidagilarini avtomatik tarzda chizib ko‘rsatadi, bunda uning foni, asosi va rangi yashirin tanlanadi. DefaultDrawing xususiyatining false qiymatini o‘rnatish hamda setka uyalarini «qo‘lda» to‘ldirish uchun mo‘ljallangan OnDrawCell voqeasi qayta ishlatgichining yozilishini talab qiladi. DefaultColWidths va DefaultRowHeights xususiyatlari yordamida yashirin tanlanayotgan barcha ustunlar va satrlarning enini o‘rnatish mumkin. ColWidth va RowHeight xususiyatlari konkret ustun enini va konkret satr bo‘yini spetsifikatsiyalaydi.

Dastur ishlash paytida siz CellRect metodi yordamida biror bir uyaning rasmini chizish uchun ma’lum sohani o‘z ixtiyoringizga olishingiz mumkin. MouseToCell metodi ustun raqami va sichqoncha kursori o‘rnatilgan satr uyasining koordinatalarini qaytarib beradi. Setkaning tanlab olingan uyasi Selection xususiyatining qiymati bo‘lib qoladi.

Dastur bajarilish paytida qaysi satr setkaning ustki satri bo‘lishini aniqlash yoki TopRow xususiyati yordamida ko‘rsatilgan satrni ustki holatga qo‘yib qo‘yish mumkin. Qaysi ustun setkaning ko‘rinib turadigan ustuni bo‘lishini aniqlash uchun LeftCol xususiyatidan foydalaning. VisibleColCount va VisibleRowCount xususiyatlarining qiymatlari setkaning ko‘rinib turgan ustunlari va satrlarining umumiy sonini spetsifikatsiyalaydi.

TImage

TImage shaklda grafik tasvir konteynerini yaratadi (bu bit obrazi, piktogramma yoki metafayla bo‘lishi mumkin).

Tasvirlar fayllari muharririning darchasi Picture xususiyati qiymatlari grafasidagi tugma bilan ochiladi. Konteyner o‘z o‘lchamlarini tasvirni to‘liq sig‘diradigan qilib o‘zgartirishi uchun AutoSize xususiyatining true qiymatini o‘rnating. Kichikroq o‘lchamdagi dastlabki tasvir butun konteynerga cho‘zilib ketishi uchun Stretch xususiyatining true qiymatini o‘rnating.

Tasvirlar fayllarining dinamik yuklanishi va saqlanishi uchun Picture obyekt xususiyatining LoadFromLile va SaveToFile metodlaridan quyidagi turlar yordamida foydalaning:

Image->Picture->LoadFromFile(«<fayl nomi>»);

Image->Picture->SaveToFile(«<fayl nomi>»);

TShape

TShape aylana va ellips, kvadrat va to‘g‘ri to‘rtburchak (burchaklarini yumaloqlash mumkin) kabi oddiy geometrik shakllarning rasmini chizadi.

Tanlab olingan geometrik shaklning turini Shape xususiyati, rang va bo‘yash usulini Brush komponentasiga joylangan ikkita Color va Style xususiyatlari aniqlaydi. Shakllarning o‘lchamlarini ham tegishli xususiyatlar aniqlaydi.

TBevel

TBevel xuddi iskana bilan o‘yilgandek hajmli ko‘rinadigan chiziqlar, bokslar yoki ramkalarni yaratadi.

Komponenta chizayotgan obyektni Shape xususiyati aniqlaydi, Style xususiyatining qiymati esa obyekt ko‘rinishini o‘zgartirib, uni bo‘rtiq yoki botiq holga keltiradi. Foydalanuvchi shakl o‘lchamlarini o‘zgartirganda ham obyektning nisbiy holatini o‘zgarmas qoldirish uchun Align xususiyatining true qiymatini o‘rnating.

TScrollBox

TScrollBox darchada o‘lchamlari o‘zgaruvchan boksni yaratadi, bu boks shu topdayoq avtomatik tarzda zaruratga ko‘ra aylantirish lineykalari bilan ta’minlanadi.

Aylantirib ko‘rish boksi yordamida darchaning ayrim sohalarini aylantirib ko‘rishdan himoyalash mumkin. Masalan, instrumentlar paneli va holat panelini himoyalash uchun avval darchani aylantirish lineykasini berkitib qo‘ying, keyin esa aylantirish boksini mijoz sohasida instrumentlar paneli va holat paneli o‘rtasida joylashtiring. Boksni aylantirib ko‘rish lineykasi darchaga tegishli bo‘lib ko‘rinadi, biroq aylantirish faqat boks ichida amalga oshiriladi.

Aylantirib ko‘rish bokslaridan yana boshqachasiga ham foydalanish mumkin: ular biror bir darchada ko‘plab aylantirib ko‘rilayotgan sohalar (turlar) yaratish imkonini beradi. Turlar ko‘pincha tijoriy matn protsessorlarida, buxgalteriya dasturlarida va loyihalarni rejalashtirish dasturlarida qo‘llanadi. Aylantirib ko‘rish boksi boshqa komponentlarga, masalan, Tbutton va TCheckBox ga ham ega bo‘lishi mumkin.

Misol: «Guruh ro‘yxati»

Dastur tavsifi

Vazifa: bir guruh o‘quvchilari haqidagi axborotni saqlaydigan va ishlay oladigan dasturni tuzish. O‘quvchi haqidagi axborotni qo‘shish yoki o‘zgartirish imkoni bilan ta’minlash.

Kerakli ko‘nikmalar

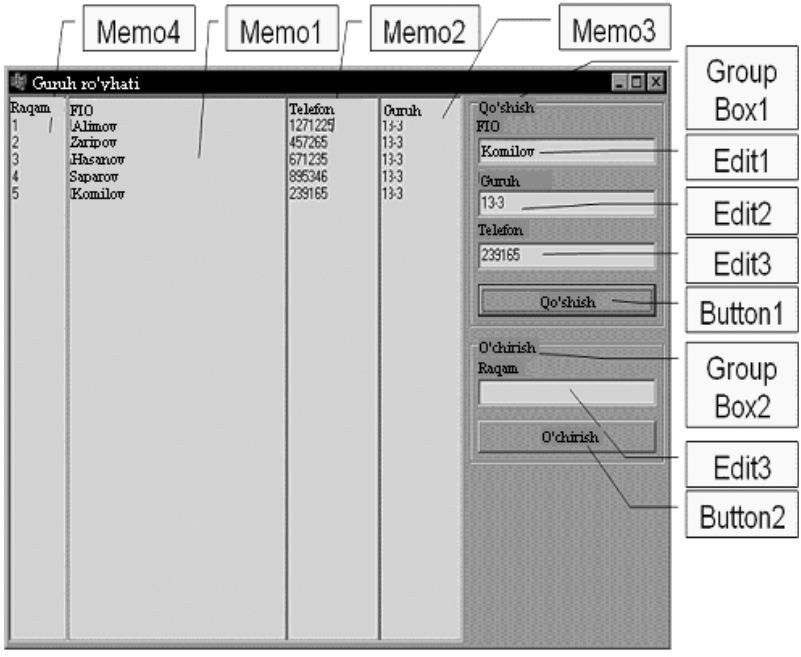
Misolni muvaffaqiyatli bajarish uchun tuzilmalarni yaxshi bilish, aqalli ularning klassik qo‘llanishini tushunish kerak. Tuzilma ma’lumotlar turidan boshqa narsa emasligini, ya’ni tuzilmalar massivlari va ularning ko‘rsatkichlarini yaratish mumkinligini anglab yetish kerak.

Yechim

Ushbu dasturni amalga oshirish uchun student tuzilmasi yaratildi. U talabalarning familiyalari va ismlarini, shuningdek, telefon va guruh raqamlarini saqlash uchun maydonlarga ega bo‘ldi. Shuningdek, o‘ttizta elementdan iborat shunday tuzilmalar massivi yaratildi. N o‘zgaruvchisi oxirgisidan keiyn keladigan element raqamiga ega. «Qo‘shimcha qilish» yoki «Yo‘q qilish» tugmalari bosilganda, N qiymati mos ravishda bittaga yo ko‘payadi, yoki kamayadi.

Shakl

Ushbu shaklda jadval ustunlari Memo maydonlarida taqdim etilgan, boshqarish elementlari GroupBox obyektlari yordamida mazmuniga ko‘ra alohida guruhlarga ajratilgan, kiritish maydonlari esa Lable obyektlari bilan belgilangan.



Dastur kodi

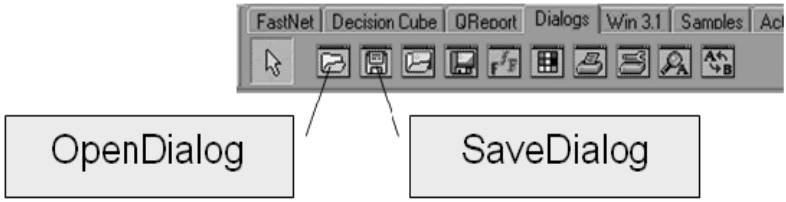
```
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------
struct student
{
AnsiString name;  // o‘quvchining familiyasi, ismi, otasining ismi
 AnsiString group;        // Guruh
int phone;                // Uy telefoni
};
```

```
student M[30];
int N = 0;
//---------------------------------------
void ShowAll()
{
 Form1->Memo1->Text = "FIO";
 Form1->Memo2->Text = "Telefon";
 Form1->Memo3->Text = "Guruh";
 Form1->Memo4->Text = "Raqam";
 for (int i = 0; i < N; i++)
 {
  Form1->Memo4->Lines->Add(IntToStr(i+1));
  Form1->Memo1->Lines->Add(M[i].name);
  Form1->Memo2->Lines->Add(IntToStr(M[i].phone));
  Form1->Memo3->Lines->Add(M[i].group);
 }
}
//---------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
ShowAll();
}
//---------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 if(N < 30)
 {
  M[N].name = Edit1->Text;
  M[N].group = Edit2->Text;
  M[N].phone = StrToInt(Edit3->Text);
   N++;
 }
 ShowAll();
}
//---------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
```

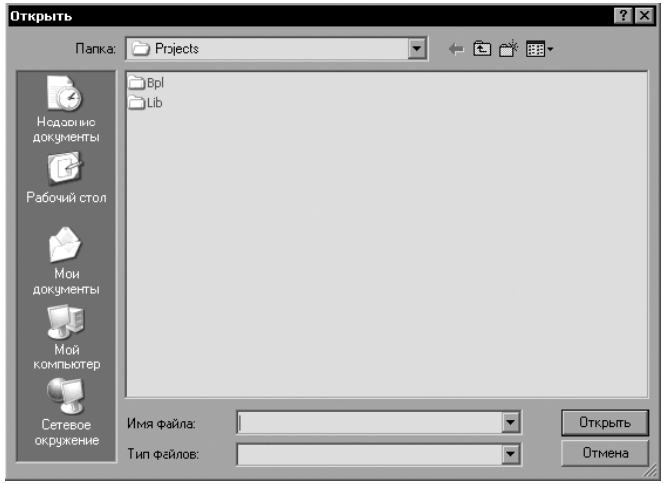
```
int j = StrToInt(Edit4->Text);
for (int i = j; i < N; i++)
{
 M[i-1] = M[i];
}
N—;
ShowAll();
}
```

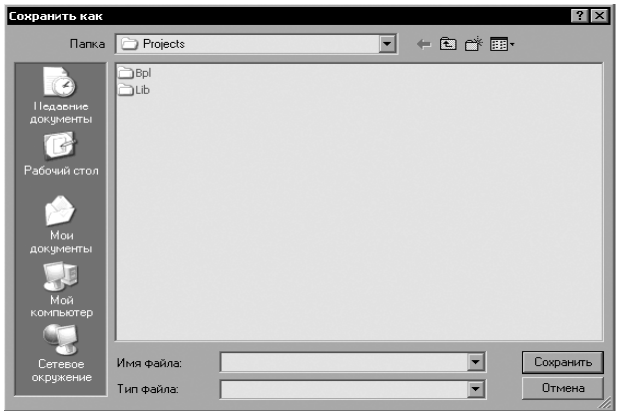
4.3. Fayllarni izlashning dialogli (ikkita individning muloqoti) darchalari

Windows operatsiya tizimida fayllarni qidirish uchun ularni saqlash va ochishning universal dialogli darchalari ko‘zda tutilgan bo‘lib, ulardan foydalanishda Dialogs qo‘shimcha ilovadan tegishli komponentalarni shaklda joylashtirish kerak:



Bu komponentalarni shaklda joylashtirgach, fayllar bilan ishlashning standart dialogli darchalarini chaqirib olish mumkin:





Misol: «Matn muharriri»

Dastur tavsifi

Vazifa: matnli fayllarni o‘zgartirish va yaratishga qodir bo‘lgan dasturni yaratish. Fayllarni diskdan ochish va kiritilgan o‘zgarishlarni saqlash imkoniyatini ta’minlash.

Fayllarni qidirish, shuningdek, faylni saqlash joyini tanlash uchun standart dialoglardan hamda fayllarni ochish/saqlashdan foydalanish.

Fayl matnini Memo maydonida aks ettirish.

Muammolar

ifstream va ofstream sinflari obyekti yaratilishda va fayl bilan assotsiatsiya qilinishda uzatilayotgan fayl nomidan belgilar massivi sifatida foydalanadi, standart dialoglar esa «satr» AnsiString turidagi qiymatlarni qaytaradi. Ya’ni ifstream yoki ofstream turdagi obyektga dialogli darcha qaytarayotgan qiymatning to‘g‘ridan-to‘g‘ri uzatilishi mumkin emas.

Bu muammoni hal qilish uchun satrni belgilar massiviga o‘zgartirib yuborish protsedurasini yaratish tavsiya qilinadi.

Zarur bo‘lgan bilimlar

Ushbu dasturni ishlab chiqish uchun ishlab chiqish muhitini standart komponentlari bilan ishlashni bilish lozim — muloqot oynalari fayllarni qidirish uchun mo‘ljallangan. Bundan tashqari, fayllarni diskda matn holatida o‘qish va saqlashni bilish lozim.

Shuningdek, fayllar matnini diskdan o‘qib olish va diskda saqlashni bilish kerak.

Yechim

Ushbu dasturni yaratishda shaklga ikkita tugmani joylashtirish kerak. Ular mos ravishda fayllarni ochish va yopish uchun mo‘l-

jallangan. Shuningdek, tegishli dialogli darchalarni ham joylashtirish kerak. Voqealar qayta ishlatgichlariga dialogli darcha chaqirishi (SaveDialog1->Execute) ni joylashtirish lozim. Dialogli darchalar voqealarining qayta ishlatgichi OnCanClosega fayllar bilan ishlashni amalga oshiradigan dasturiy kodni joylashtirish kerak.

Fayllar bilan ishlash dialogli darchaning OnCanClose voqeasi yuzaga kelganda, tegishli dialogli darchaning FileName xususiyatida tanlangan fayl nomi bo‘ladi.

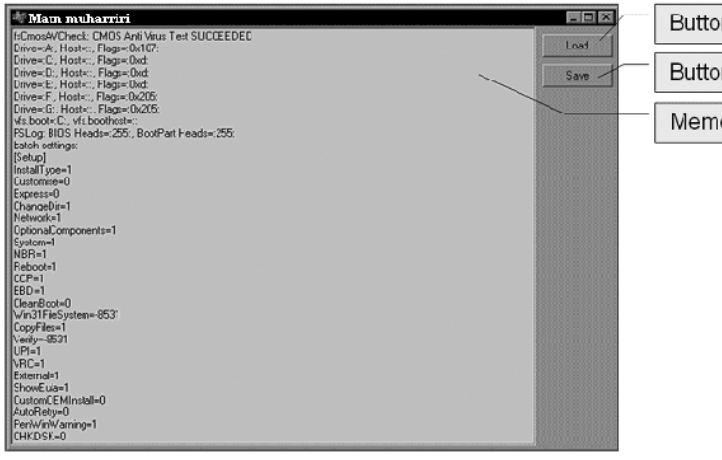
ifstream sinfi obyektining satriga yozilgan fayl haqidagi axborotni uzatish uchun satrni belgilar massivida qayta o‘zgartirish kerak. Buning hammadan oson yo‘li — massivning birinchi elementiga iqtibosni uzatadigan protsedurani yaratish. Bu protsedura muntazam ravishda, elementma-element, satrdan belgilarni olgan holda massivni to‘ldirishi kerak. Bu protsedura yordamida barcha zarur qayta o‘zgarishlarni osongina amalga oshirish mumkin.

Fayl ichidagini Memo1 maydoniga yozish uchun ifstream sinfining getline() funksiyasi yordamida satrlarni izchil o‘qib borish hamda ularni Memo1 maydoniga, bu maydonning tarmoq obyekti Line(Memo1->Lines->Add(satr);) ga tegishli Add() funksiyasi yordamida yozib qo‘yish kerak.

Faylda axborotni saqlash uchun unga Memo1 obyekti satrlarini belgima-belgi yozib qo‘yish kerak, bunda yangi satrni satr oxiri belgisi (\n) bilan boshlash kerak.

Shakl

Formada ko‘rinib turadigan komponentalardan tashqari, shuningdek, Dialogs - SaveDialog va LoadDialog qo‘shimcha ilovalardan olingan komponentalar ham mavjuddir.



Dastur kodi

```
#include <vcl.h>
#include <fstream.h>

#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------
void AnsiStringToCharPointer(char * c, AnsiString s, int n = 0)
{
 int k = s.Length();
 if(k > n && n != 0){ k = n; }
 for(int i = 0; i < k; i++)
 {
  c[i] = s[i+1];
 }
}
//---------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 OpenDialog1->Execute();
}
//---------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
SaveDialog1->Execute();
}
//---------------------------------------------------
void __fastcall TForm1::OpenDialog1CanClose(TObject *Sender,
      bool &CanClose)
{
 AnsiString name = OpenDialog1->FileName;
```

```
 char c[100] = "";
 AnsiStringToCharPointer(c,name);
 ifstream in(c);
 Memo1->Text = "";
 while(in.good())
 {
  in.getline(c,100);
  Memo1->Lines->Add(c);
 }
 in.close();
}
//---------------------------------------
void __fastcall TForm1::SaveDialog1CanClose(TObject *Sender,
      bool &CanClose)
{
 AnsiString name = SaveDialog1->FileName;
 char c[1000] = "";
 AnsiStringToCharPointer(c,name);
 ofstream out(c);
 for(int i = 1;i < Memo1->Lines->Count;i++)
 {
  AnsiString s = Memo1->Lines->operator [](i);
  for(int j = 1; j < s.Length()+1; j++)
  {
   cout << s[j];
  }
  cout << '\n';
 }
 cout.close();
}
```

4.4.Grafika

C++Builder da grafikani qo‘llab-quvvatlash

C++Builder dasturi Windows GDI funksiyalarini turli darajalarda inkapsulatsiyalaydi. Bu o‘rinda bir usul muhim bo‘lib, uning vositasida grafik komponentalar o‘z tasvirlarini monitor ekranida taqdim etadi. GDI funksiyasi to‘g‘ridan-to‘g‘ri chaqirilganda, ushbu grafik komponentalarga *qurilma konteksti deskriptori (device context handle)* ni uzatish kerak. Bu deskriptor siz tanlab olgan rassomchilik ashyolari — perolar, mo‘yqalamlar, shriftlarni chiqarib beradi. Grafik tasvirlar

bilan ishlash tugagach, siz qurilma kontekstini dastlabki holatga keltirib qo‘yishga majbursiz va shundan keyingina undan ozod bo‘lishingiz mumkin.

Shu darajada detallashtirilgan grafika bilan ishlashga sizni majbur qilish o‘rniga, C++Builder grafik komponentalarning Canvas (Kanva — Asos) xususiyati vositasida sodda va tugal interfeysni taklif qiladi. Bu xususiyat qurilmaning to‘g‘ri kontekstini nomlaydi (initsiallashtiradi) hamda siz rasm chizishni to‘xtatgan kerakli vaqtda uni ozod qiladi. Asos pero, mo‘yqalam va shrift tavsiflari nomidan ish ko‘radigan berilgan xususiyatlarga ega.

Grafik komponentalar bilan ishlashda foydalanuvchi amalga oshirishi lozim bo‘lgan yagona ish — bu qo‘llanayotgan rasm chizish ashyolarining tavsiflarini aniqlash. Ashyolarni yaratish, tanlash va ozod qilishda sizdan tizim zaxiralarini kuzatib borish talab qilinmaydi. Asosning o‘zi bu haqda qayg‘uradi.

Grafika bilan ishlashda C++Builder namoyon qiladigan afzalliklardan biri — bu tizimning grafik zaxiralari uchun keshlangan xotiradan foydalanish. Aytaylik, agar sizning dasturingiz biror bir konkret turdagi peroni qaytadan yaratsa, qo‘llasa va ozod etsa, siz ushbu perodan har gal foydalanganingizda bu qadamlarni takrorlashingizga to‘g‘ri keladi. C++Builder kesh-xotiradan grafik zaxiralarni saqlash uchun foydalanar ekan, tez-tez qo‘llanadigan rasm chizish ashyosi har gal yangitdan qayta yaratilmay, balki kesh-xotiradan takroran tanlab olinishi ehtimoli oshadi. Buning natijasida sizning grafik ilovangizning takrorlanayotgan operatsiyalarining samarasi ancha ortishi aniq.

Quyida Windows uchun ilovalar darchasida ko‘k kontur bilan aylantirilgan sariq ellipsni chizish masalasini hal qilayotgan kod fragmenti keltirilgan. Bu masala rasm chizish asosi vositasida yechiladi.

```
void_fastcall TFormPaint(TObject*Sender) {
Canvas->Pen->Color=clBlue;  // Canvas->Brush->
Color=clYellow konturining rangini tanlash; // Canvas->Ellipse(10,
20, 50, 50) ichining rangini tanlash; // ellips rasmini chizish}
```

Asosdan foydalanish

Asosning obyektli sinfi Windows ning grafik funksiyalarini, alohida chiziqlar, shakllar va matnni chizish uchun mo‘ljallangan yuqori darajadagi funksiyalardan boshlab inkapsulatsiyalaydi. Keyin rasm chizish uchun asos bilan ishlaydigan o‘rta darajadagi usul va metodlar

keladi. Nihoyat, quyi darajada Windows GDI funksiyalarining o‘ziga kirish ta’minlanadi. Quyidagi jadvalda asosning asosiy metod va usullarining umumlashtirilgan tavsiflari berilgan:

Daraja	Xatti-harakatlar	Metodlar	Xususiyatlar
Yuqori	Peroning joriy pozitsiyasini belgilaydi	MoveTo	PenPos
	To‘g‘ri chiziqni berilgan nuqtagacha chizadi		
	Berilgan o‘lchamdagi to‘g‘ri to‘rtburchak chizadi		
	Berilgan o‘lchamda ellips chizadi		
	Matn satrini chizadi		
	Matn satrini chiqarish uchun ajratilgan bo‘yni beradi		
	Matn satrini chiqarish uchun ajratilgan enni beradi		
	To‘rtburchak ichida matn chiqarish		
	Ko‘rsatilgan to‘rtburchak ichiga rang va joriy mo‘yqalam teksturasini quyish		
	Asos sohasiga (ixtiyoriy shakldagi) berilgan rang quyish		
O‘rta	Peroning rangi, uslubi, eni va rejimini o‘rnatish uchun qo‘llanadi		
	Grafik shakllar va asos fonini quyishda rang va teksturani o‘rnatish uchun qo‘llanadi		
	Berilgan rang, o‘lcham va uslubdagi shriftni o‘rnatish uchun qo‘llanadi		
	Berilgan asos pikselini o‘qish va rangini yozish uchun qo‘llanadi		
	CopyMode rejimida asosning to‘rtburchak sohasidan nusxa ko‘chiradi		
	Asosning to‘rtburchak sohasidan rangni almashtirib nusxa ko‘chiradi		
	Asosning berilgan joyida bit obrazini, piktogramma va metafaylning rasmini chizadi		
	Bit obrazini, piktogramma va metafaylning rasmini to‘rtburchakni to‘liq to‘ldiradigan qilib chizadi		
Quyi	Windows GDI funksiyalarini chaqirishda parametr sifatida qo‘llanadi		

Rasmlar bilan ishlash

C++Builder muhitida bajariladigan grafik ishlarning asosiy mundarijasi asosingiz shaklida yoki unda joylashtirilgan boshqa komponentalarda bevosita rasm chizishdan iborat. C++Builder,

shuningdek, tashqi tasvirlar — bit obrazlari, metafayllar, piktogrammalar, shu jumladan, palitralarni avtomatik boshqarishning qo‘llab-quvvatlanishiga xizmat ko‘rsatadi.

C++Builder muhitida rasmlar bilan ishlashda uchta muhim jihatni e’tiborga olish kerak.

Rasm, grafika yoki asos

C++Builder muhitida grafikaga aloqador bo‘lgan uch xil obyekt mavjud:

— **Asos** shaklda, grafik komponentada, printerda yoki boshqa biron bit obrazida rasm chizish uchun mo‘ljallangan yuzaning bit kartasidan iborat. Asos mustaqil obyekt emas, u hammavaqt biror bir boshqa grafik obyektning xususiyati bo‘lib xizmat qiladi.

— **Grafika** biror bir fayl yoki zaxiraning (bit obrazi, piktogramma yoki metafayl) rastrli tasviridan iborat. C++Builder dasturi TGraphic bazaviy sinfning hosilalari bo‘lgan TVitmap, Ticon va TMetafile obyektli sinflarini aniqlaydi. Albatta, siz o‘zingiz grafik obyektlarning shaxsiy sinflarini e’lon qilishingiz mumkin. Tgraphic sizning ilovangizda grafikaning barcha turlaridan foydalanish uchun minimal standart interfeys taqdim etadi.

— **Rasm** — grafika uchun konteyner bo‘lib, u grafik obyektlarning har qanday sinflarini taqdim etadi.Shunday qilib, TRicture konteynerli sinf bit obrazi, piktogramma, metafayl yoki foydalanuvchi tomonidan belgilangan boshqa biron grafik turga ega bo‘lishi mumkin. Ilova esa «rasm» obyekti vositasida konteynerning barcha obyektlariga standartlashgan tarzda murojaat qilishi mumkin. Darhaqiqat, tasvirlarni boshqarish komponentalarining ko‘pchiligi obyektli turdagi Tricture ning Picture xususiyatiga ega bo‘lib, u har xil turdagi grafik tasvirlarni taqdim etish imkoniyatiga ega.

Shuni ham ta’kidlab o‘taylik: «rasm» obyekti hammavaqt biror bir grafikaga ega bo‘lib, bu grafika, o‘z navbatida, asosga ega bo‘lishga ehtiyoj sezadi (asosga ega bo‘lgan yagona standart grafik sinf — bu TVitmap dir). Odatda, rasm bilan ishlar ekansiz, siz grafik obyektning faqat TRicture konteyneri orqali kirish uchun ochiq bo‘lgan qismi bilangina ish ko‘rasiz. Agar sizga konkret grafik obyektga kirish huquqini ko‘rsatish lozim bo‘lib qolsa, ushbu rasmning Graphic xususiyatiga murojaat qiling.

Grafik fayllar

Ilovangiz ishlayotgan har bir daqiqada C++Builder standart formatdagi tasvirlar fayllarida rasmlar va grafiklarning yuklanishi va

saqlanishini qo‘llab-quvvatlaydi. Saqlanayotgan va dastlabki faylning nomlari bir-biriga mos kelishi hamda bir-biridan farq qilishi mumkin.

Tasvirni fayldan rasmga yuklatish uchun LoadFromFile rasm metodidan foydalaning. Faylda tasvirni saqlash uchun rasm metodidan foydalaning. Ushbu metodlarning yagona parametri bu fayl nomi bo‘ladi. LoadFromFile metodi fayl kengayishidan o‘zi yaratadigan va saqlaydigan grafik obyekt turini aniqlash uchun foydalanadi. LoadToFile metodi saqlanayotgan grafik obyekt turiga mos keladigan kengayishli faylni saqlaydi: LoadFromFile

Navbatdagi matnda yo‘riqnoma (instruksiya) mujassam bo‘lib, uni siz bit obrazini komponentali rasm obyektiga yuklash uchun kod moduli matniga yozib qo‘yishingiz kerak:

```
void_fastcall TFormI::FormCreate(TObject *Sender) {
Imagel->Picture>LoadFromFile(″c:\\windows\\clouds.bmp″);
}
```

Rasm .bmp bit obrazlari fayllarining standart kengayishini tanib oladi hamda o‘z grafikasini TVitmap sinf obyekti sifatida yaratadi, keyin esa ko‘rsatilgan ismli fayldan tasvirni yuklash LoadFromFile metodini chaqiradi.

Palitraga xizmat ko‘rsatish

Foydalanuvchilik interfeysining ko‘pchilik elementlari biror bir palitraga muhtojlik sezmaydi. Biroq, grafik tasvirlarga ega komponentalarga, komponentalar ma’lumotlarini tegishli tarzda aks ettirish uchun, Windows hamda uning ekran drayveri bilan o‘zaro aloqaga kirishish zarur bo‘lib qolishi mumkin. Windows operatsiya tizimiga oid hujjatlarda bu jarayon *palitralarni ishga tushirish (palett realizing)* deb ataladi. Palitrani ishga tushirish operatsiyasining vazifasi shundan iboratki, u eng ustki (ekranda sizga nisbatan eng yaqin turgan) faol darcha to‘liq rang palitrasidan foydalanishini, fon darchalari esa o‘z palitralarining qolgan ranglaridan maksimal darajada foydalanishlarini ta’minlashi kerak. Bu degani, fon darchalari o‘z ranglarini «real» palitradagi erishish mumkin bo‘lgan eng yaqin ranglarga o‘zgartira olishlari kerak. Darchalar bir-birini qisman yopib joy almashar ekan, Windows ham muttasil darcha palitralarini ishga solib boradi.

Mulohaza. C++Builder bit obrazlari palitralaridan boshqa palitralarni yaratish va ularga xizmat ko‘rsatish uchun mustaqil vositalarga ega emas. Biroq, agar siz biror bir palitraning deskriptorini olgan bo‘lsangiz, grafik komponentalar ular bilan ishlay oladi.

Displey yoki printer turidagi qurilmalar bilan ishlashda C++Builder komponentalari avtomatik tarzda palitralarni ishga tushirish mexanizmini qo‘llab-quvvatlaydi. Shunday qilib, siz TControl bazaviy komponentli sinfdan meros qilib olingan ikkita GetPalette va PaletteChanged metodlaridan foydalanishingiz mumkin. Bunda Windows bu palitraga qanday munosabatda bo‘lsa, siz ham uni xuddi shunday ishlata olasiz.

— **Palitraning komponenta bilan aloqasi.** Agar grafik komponenta uchun biror bir palitradan foydalanish zarurati tug‘ilgan bo‘lsa, sizning ilovangiz bu haqda xabardor bo‘lishi kerak. Palitrani komponentangizga o‘xshatish uchun uning GetPalette obyektli metodini shunday ortiqcha yuklatingki, u ushbu palitra *deskriptori (handle)* ni qaytarsin. Shuning bilan birga siz, birinchidan, komponentangizning ma’lum bir palitrasi ishga tushishi lozimligini ilovangizga ma’lum qilasiz, ikkinchidan, ishga tushishda qaysi palitra konkret qo‘llanishi kerakligini aniqlaysiz.

— **Palitra o‘zgarishiga reaksiya (munosabat).** Sizning komponentangiz GetPalette metodini ortiqcha yuklatish vositasida qandaydir palitra bilan o‘xshatilgan bo‘lsa, C++Builder tizimi PaletteChanged metodi yordamida Windows ning palitralardan xabarlariga munosabat bildirishni avtomatik tarzda o‘z zimmasiga oladi. Normal ish sharoitida siz hech qachon yashirin belgilangan bu metodning xulq-atvorini qayta aniqlash zaruratiga duch kelmaysiz. PoletteChanged metodining asosiy vazifasi palitrani ishga tushirish turini (fonli yoki faol darchalar uchun) aniqlashdan iborat. Palitralarning Windows tizimida ishga tushirilishiga nisbatan C++Builder bir qadam ilgarilab ketdi: darcha deskriptorlari yordamida, nafaqat bir-birining ustiga taxlanadigan «dasta» palitrasi, balki faol darchaning bir-birining ustiga taxlangan komponentalarining palitralari ham ishga tushiriladi. Agar xohlasangiz, siz palitralarning yashirin qabul qilingan bunday xulq-atvorini qayta aniqlashingiz va natijada biror bir komponenta to‘liq rang palitrasiga ega bo‘lishi hamda ekranda sizga eng yaqin turgan komponentadek ko‘rinishiga erishishingiz mumkin.

Ekrandan tashqaridagi bit obrazlari

Windows uchun murakkab grafik ilovalarni dasturlashning umum qabul qilingan metodikasi shundan iboratki, bunda ekrandan tashqari bit obrazi yaratiladi, bu obrazga konkret tasvir tushiriladi yoki to‘ldiriladi va, nihoyat, yaratilgan tasvir to‘laligicha bit obrazidan ekran darchasining ko‘rsatilgan joyiga nusxa ko‘chirib olinadi. Natijada ekran darchasida bevosita takroran rasm chizish keltirib chiqaradigan va ko‘zni charchatadigan monitor ekranidagi lipillashlar kamayadi.

C++Builder sizning ilovangizda Tbitmap sinfi obyektlarini yaratish imkonini beradi, bu ekrandan tashqari tasvirlar sifatida ishlay oladigan fayl va boshqa zaxiralar tasvirlarini ham sizning ilovangizda taqdim etish uchun qilinadi.

Bit obrazlaridan nusxa ko‘chirish

C++Builder bir asosdan ikkinchisiga nusxa ko‘chirishning to‘rtta usulini ko‘zda tutadi. Erishish zarur bo‘lgan natijaga muvofiq keladigan kerakli nusxa ko‘chirish usulini quyidagi jadvaldan tanlab oling:

Talab qilingan natija	Metod
Grafikadan to‘liq nusxa ko‘chirish	Draw
Masshtablab nusxa ko‘chirish	StretchDraw
Asosning to‘rtburchak uchastkasidan nusxa ko‘chirish	CopyRect
Rastrli operatsiyalar bilan nusxa ko‘chirish	BrashCopy

Yaratish va xizmat ko‘rsatish

Murakkab grafik tasvirlarni yaratishda ularni ilovangizning ekran darchasida turgan shakl yoki komponenta asosida bevosita chizishdan qochish kerak. Buning o‘rniga siz biror bir bit obrazi obyektining konstruksiyasini yaratishingiz va mana shuning asosiga rasm chizishingiz, keyin esa uni ekran asosiga nusxa olib ko‘chirishingiz mumkin. Ekrandan tashqaridagi bit obrazi asosiga rasm chizishda grafik komponentalarning Paint metodi ko‘proq qo‘llanadi.

Ekrandan tashqari bit obrazida murakkab tasvirni yaratishga bag‘ishlangan misol namunasini Palitralar komponentasining Samples qo‘shimcha ilovasida keltirilgan (Gauge) indikatorida ko‘rishimiz mumkin. Tgauge komponentasi dasturiy modulining Gauges.cpp va Gauges.h dastlabki fayllarini \...\CBuilder\Examples\Control\Source katalogidan topish mumkin.

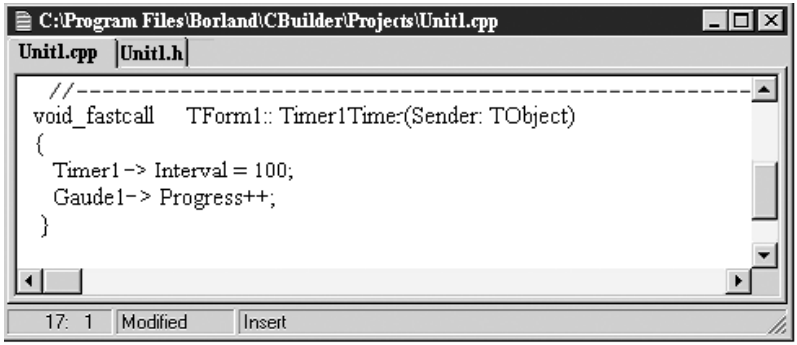
Gauges.cpp faylining fragmenti ekrandan tashqari TBitmap bit obrazining asosida Paint rasm chizish komponentli metod bajaradigan asosiy operatsiyalarni ko‘rsatib beradi. Avval PaintBaskgroud funksiyasi Color dan olingan rang bilan indikatorning to‘rtburchak fon sohasini bo‘yab chiqadi. Keyin King xususiyatining berilgan qiymatiga muvofiq kerakli shakl konturi ForeColor xususiyatli rang bilan aylantirib chiqiladi hamda ichiga BackColor xususiyatli rang quyiladi (strelkali King=gkNeedle indikatori qo‘llangan bizning misolimizda bu ishni PaintAsNeedle funksiyasi bajaradi). So‘nggi instruksiyalar CopyMode asosining nusxa ko‘chirish rejimi xususiyatlarini belgilaydi, indikatorni matn bilan ta’minlaydi (PaintAsText metodi bilan) va faqat shundan

keyingina (Draw metodi yordamida) ekrandan tashqari bit obrazining asosi ekranda aks ettiriladi:

```
void _fastcall TGauge::Paint()
{
std::auto_ptr<Graphics::TBitmap> Thelmage
 (new Graphics::TBitmap() ) ;
std::auto_ptr<TBltBitmap> Overlaylmage (new TBitBitmap());
TRect PaintTRect;
The Image->Height = Height;
TheImage->Width = Width;
PaintBackgroundtThelmage.get()) ;
PaintTRect = ClientRect;
 if (FBorderStyle == bsSingle)
InflateRect(&RECT(PaintTRect), -1, -1);
OverlayImage->MakeLike(Thelmage.get() ) ;
PaintBackground(Overlay Image.get());
switch(FKind) {
case gkText:
PaintAsNothing(Overlay Image.get(), PaintTRect); break;
case gkHorizontalBar:
 case gkVerticalBar:
PaintAsBar(Overlaylmage.get(), PaintTRect); break;
case gkPie:
PaintAsPie(Overlay Image.get(), PaintTRect); break;
case gkNeedle:
PaintAsNeedle(Overlaylmage.get(), PaintTRect); break;
}
 The Image->Canvas->CopyMode = cmSrcInvert;
TheImage->Canvas->Draw(0, 0, Overlaylmage.get()) ;
 TheImage->Canvas->CopyMode = cmSrcCopy;
if (ShowText == true)
PaintAsText(Thelmage.get(), PaintTRect);
Canvas->CopyMode = cmSrcCopy;
Canvas->Draw(0, 0, Thelmage.get()) ;
}
```

Indikator millari (strelkalari)ning taymer signallari bo‘yicha harakatlanishini ishga soladigan dastur matni System qo‘shimcha ilovasiga kiradigan TTimer komponentasi OnTimer voqealar qayta ishlagichining faqat ikkita satridan iborat ekanini 4.18-rasmdan

ko‘rishimiz mumkin. Voqeani qayta ishlovchi funksiya Timer1Timer ning birinchi instruksiyasi Timer1 obyektining bir daqiqali vaqt intervali uchun 1000 qiymatini (yashirin qabul qilingan) o‘rnatadi. Ikkinchi instruksiya esa Gauge1 obyektining Progress xususiyati qiymatini inkrementatsiya qiladi.



4.18-rasm. Milli (strelkali) shkala ko‘rinishidagi indikator.

Agar siz ushbu ilovani yig‘ib, ishga tushirib yuborsangiz, indikator millari qanday tezlikda harakatlanmasin, monitor ekranida lipillash baribir sodir bo‘lishiga ishonch hosil qilasiz.

O‘zgarishlarga reaksiya (munosabat)

Barcha grafik obyektlar (shu jumladan, rasm chizish uchun asoslar) hamda ular egalik qilgan obyektlar (perolar, mo‘yqalamlar va shriftlar) obyektda sodir bo‘lgan o‘zgarishlar uchun javob beradigan qurilma voqealarga ega bo‘ladi. Bu voqealar yordamida siz o‘z komponentlaringizni va, demakki, ulardan foydalanuvchi ilovalarni yuz bergan o‘zgarishlarga javoban o‘z tasvirlaringizni qayta chizishga majbur qila olasiz.

Agar bu obyektlar komponenta modulining dastlabki faylida punlished sifatida e’lon qilingan bo‘lsa, grafik komponentaning o‘zgarishlariga munosabat bildirilishi (reaksiya), ayniqsa, muhimdir. Bu holda ilovani loyihalash bosqichida komponenta turi obyektlar Noziri o‘rnatgan xususiyatlarga muvofiq kelishini ta’minlashning yagona usuli komponentli obyekt o‘zgarishlariga munosabat bildiradigan OnChange voqealar qayta ishlatgichini ulashdan iborat.

```
class TMyShape : public TGraphicControl
{
public:
 virtual _fastcall TMyShape(TComponent* Owner);
```

```
__publi shed:
TPen *FPen;
TBrush *FBrush;
 void_fastcall StyleChanged(TObject *Sender) ;
};
 _fastcall TMyShape::TMyShape(TComponent* Owner)
: TGraphicControl(Owner) {
Width = 64;
Height = 64;
PPen = new TPen;
FPen->OnChange = StyleChanged; // Pero uslubi o‘zgartirilsin
FBrush = new TBrush;
FBrush->OnChange = StyleChanged; // mo‘yqalam uslubi
o‘zgartirilsin }
 void_fastcall TMyShape::StyleChanged(TObject *Sender) (
Invalidate();
}
```

Vizual Komponentalar Kutubxonasining geometrik shakllarni chizish TShape grafik komponentasi o‘zining pero va mo‘yqalamdan iborat xususiyatlarini **_published** seksiyasida e’lon qiladi. Komponenta obyektining konstruktori StyleChanged metodini OnChange voqeasiga taqdim etadi hamda buning bilan komponentani o‘zida tasvirlangan shakllarni pero va mo‘yqalamning har qanday o‘zgarishlarida ham qayta chizishga majbur etadi.

Misol: «Yuza».

Dastur tavsifi

Bu dasturga berilgan vazifa quyidagicha: ekranga tasodifiy yuzaning to‘rtburchak shakliga yaqinlashtirilgan aksini tushirib, ikki o‘lchamli massiv yaratish. Bu massiv yuza Z koordinatasining nuqtada tasodifiy qiymatlariga ega bo‘ladi, massivning qolgan koordinatalari massiv indekslari bilan aniqlanadi.

Muammolar

1.Tasodifiy shakldagi to‘rtburchakni ekranga chiqarish uchun shaklga piksellarning bevosita chiqarilishidan foydalanish kerak. Bu muammoni yechish variantlaridan biri — ko‘pburchak shaklni chiqarish uchun funksiyalar to‘plamiga ega bo‘lgan shaklning Canvas xususiyatiga murojaat qilish.

Tasodifiy ko‘pburchak shaklni ekranga chiqarish uchun (Form1-> Canvas->Poligon(....)) shakli Canvas obyektining Polygon protsedurasidan foydalanish mumkin. Bu protseduraga ko‘pburchakli shakl burchaklarining soni va TPoint turidagi obyektlar massivi (u o‘z ichida int — X va Y turidagi ikkita o‘zgaruvchiga ega) olib beriladi.

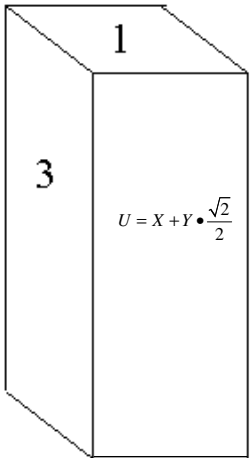
Shunday qilib, ushbu protseduradan foydalanish uchun avval o‘xshash obyektlar massivini yaratish hamda har bir nuqta uchun X va Y qiymatlarini berish kerak. Ma’lum koordinataning nuqta uchun qiymati <nuqta nomi>, <koordinataning nomlanishi> konstruksiyasi yordamida beriladi.

Misol:
TPoint T;
T.x=100;
T.y=117

2.Uch o‘lchamli tasvirni rag‘batlantirish uchun yuza mavjud bo‘lgan fazo koordinatalarini yassi ekran koordinatalariga qayta o‘zgartirish kerak. Bu muammoni yechish uchun koordinatalarni qayta o‘zgartiradigan oddiy matematik formulalarni jalb qilish lozim.

Aytaylik, yuza mavjud bo‘lgan fazo koordinatalari X, Y, Z, monitor koordinatalari esa U va V bo‘lsin. Bu holda koordinatalarni qayta o‘zgartirish formulalari quyidagi ko‘rinishda bo‘ladi:

$$U = X + Y \bullet \frac{\sqrt{2}}{2} \qquad V = Y + \frac{Z}{ConstH}$$

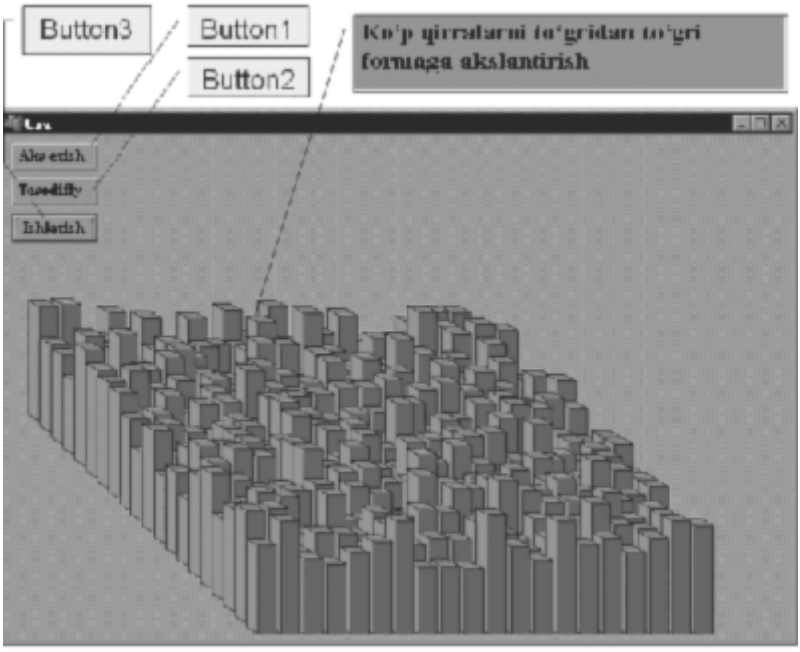


Shakl

Bu yerda ConstH — gorizontal va vertikal o‘lchamlarni bog‘laydigan qandaydir konstanta.

3.Ko‘rgazmalilikni oshirish uchun yuzani approksimatsiya qiladigan elementlarni parallelopipedlar ko‘rinishida taqdim etish kerak, buning uchun ular odatda turli rangdagi uchta ko‘pburchak sifatida tasvirlanadi. Aytaylik, keltirilgan formulalar bo‘yicha biz hisoblab chiqqan koordinatalar parallelopiped burchaklaridan birini aniqlaydi. Qolgan burchaklar koordinatalarini bosh burchak koordinatalariga shunchaki bir nechta piksellarni qo‘shib, hisoblab chiqish qiyinchilik tug‘dirmaydi.

Ushbu dasturda shakl bevosita grafik chiqarish uchun maydon sifatida qo‘llanadi. Shuning uchun u faqat boshqarish tugmalariga ega bo‘lib, ular o‘z vazifalariga ko‘ra yuzani aks ettirish, yangi tasodifiy koordinatalarini tanlash funksiyalari hamda avvalgi misollardagi tamoyillar bo‘yicha ishlaydigan ″Ishlatish/To‘xtatish″ tugmasini eslatadi:



Dastur kodi:

```
#include <vcl.h>
#pragma hdrstop
#include «Unit1.h»
//———————————————————————————
#pragma package(smart_init)
#pragma resource ″*.dfm″
TForm1 *Form1;
//———————————————————————————
int Pole[20][20];       // Pole
//———————————————————————————
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
```

```
}
//———————————————————————————————————
void DrawPole()
{
 TPoint P[4];
 for(int y = 0; y < 20; y++)
  for(int x = 19; x >= 0; x—)
  {
   P[0].x = x * 20 + y * 10  + 22;
   P[0].y = y * 10 - Pole[x][y] + 202;
   P[1].x = P[0].x + 16;
   P[1].y = P[0].y;
   P[2].x = P[0].x + 26;
   P[2].y = P[0].y + 6;
   P[3].x = P[0].x + 10;
   P[3].y = P[0].y + 6;
   TPoint P2[4] = {P[0],P[1],P[2],P[3]};
   TPoint P1[4] = {P[0],P[1],P[2],P[3]};
   P2[1].y = y * 10 + 256;
   P2[1].x = P[2].x;
   P2[0].y = y * 10 + 256;
   P2[0].x = P[3].x;
   P1[1].x = P[0].x;
   P1[1].y = y * 10 + 250;
   P1[2].x = P[3].x;
   P1[2].y = y*10 + 256;
   Form1->Canvas->Brush->Color = clBlue;
   Form1->Canvas->Polygon(P2,3);
   Form1->Canvas->Brush->Color = clRed;
   Form1->Canvas->Polygon(P1,3);
   Form1->Canvas->Brush->Color = clGreen;
   Form1->Canvas->Polygon(P,3);
  }
}
//———————————————————————————————————
void RandomPole()
{
 for(int y = 0; y < 20; y++)
  for(int x = 0; x < 20; x++)
  {
```

```
   Pole[x][y] = random(50)+10;
  }
}
//---------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
 for(int i = 0; i < 20; i++)
  for(int j = 0; j < 20; j++)
  {
   Pole[i][j] = 0;
  }
}
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 Form1->Refresh();
 DrawPole();
}
//---------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
 RandomPole();
}
//---------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
 if(Button3->Caption == "To‘xtatish")
 {Button3->Caption = "Ishlatish";}
 else
 {Button3->Caption = "To‘xtatish";}
}
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
 if(Button3->Caption == "To‘xtatish")
 {
  RandomPole();
  Form1->Refresh();
  DrawPole();  }}
```

FOYDALANILGAN ADABIYOTLAR

1. **Гради Буч.** Объектно-ориентированный анализ и проектирование с примерами приложений на C++. Невский диалект. 2001.

2. **И. Грехем.** Объектно-ориентированные методы. Принципы и практика. Вильямс. 2004.

3. **Г.С.Иванова.** Объектно-ориентированное программирование. Учебник. МГТУ им. Баумана. 2003.

4. **М.Фаулер, К.Скотт.** UML в кратком изложении. Применение стандартного языка объектного моделирования. М. Мир. 1999.

5. **Г.Буч, Д. Рамбо, А. Джекобсон.** Язык UML: руководство пользователя. М. ДМК. 2000.

6. **Пол Айра.** Объектно-ориентированное программирование на C++. Второе издание. М. Бином. 1999.

7. **В.В.Подбельский.** Язык C++. М. Финансы и статистика. 1996.

8. **Б. Страуструп.** Язык программирования C++. Третье издание. М. Бином. 1999.

9. **Д. Либерти.** Освой самостоятельно C++: 10 минут на урок. Пер с англ. Вильямс. 2004.

10. **Я.К.Шмидский.** Программирование на языке C++. Самоучитель. Учебное пособие. Диалектика. 2004.

11. **А. Б. Крупник.** Изучаем C++. Питер. 2003.

12. **С. Мейерс.** Наиболее эффективное использование C++. 35 новых рекомендаций. ДМК-Пресс. 2000.

13. **Т. Фейсон.** Объектно-ориентированное программирование на C++ 4.5. Киев. Диалектика. 1996.

14. **Г. Шилдт.** Самоучитель C++. Второе издание. СПб. BHV. 1998.

15. **Г. Шилдт.** Теория и практика C++. СПб. BHV. 1996.

16. **Дж. Элджер.** C++: библиотека программиста. СПб. Питер. 1999.
17. **Д.В. Николаенко.** Самоучитель по Visual C++.СПб. 2001.
18. **П. Киммел.** Borland C++5. СПб. BHV. 1997.
19. **К. Грегори.** Использование Visual C++6. М. Вильямс. 1999.
20. **Крейг Арнуш.** Borland C++: освой самостоятельно. М. Бином. 1997.
21. **Р. Лейнекер.** Энциклопедия Visual C++6. СПб. Питер. 1999.
22. **Н.Ю.Секунов.** Самоучитель Visual C++6. СПб. BHV. 1999.
23. **К. Паппас., У. Мюррей.** Visual C++6: Руководство разработчика. Киев. BHV. 2000.

MUNDARIJA

Sh.A. Nazirov, R.V. Qobulov

OBYEKTGA MO‘LJALLANGAN DASTURLASH

Kasb-hunar kollejlari uchun o‘quv qo‘llanma

Muharrir *Ma’suda Yo‘ldosheva*
Musavvir *Anatoliy Bobrov*
Badiiy muharrir *Rustam Zufarov*
Texnik muharrir *Tatyana Smirnova*
Musahhih *Dono To‘ychiyeva*
Kompyuterda sahifalovchi *Akmal Sulaymonov*

IB № 4491

Bosishga 22.06.2007-y.da ruxsat etildi. Bichimi 60x90$^{1}/_{32}$.
Tayms garniturasi. Ofset bosma. 11,55 shartli bosma toboq.
12,5 nashr tobog‘i. Jami 6528 nusxa. raqamli buyurtma.
8-2007 raqamli shartnoma. Bahosi shartnoma asosida.

O‘zbekiston Matbuot va axborot agentligining
G‘afur G‘ulom nomidagi nashriyot-matbaa ijodiy uyi.
100129. Toshkent, Navoiy ko‘chasi, 30.
100128. Toshkent, Usmon Yusupov ko‘chasi, 86.

Bizning internet manzilimiz: www.iptdgulom.uz