O’ZBEKISTON RESPUBLIKASI AXBOROT TEXNOLOGIYALARI VA KOMMUNIKATSIYALARINI RIVOJLANTIRISH VAZIRLIGI

TOSHKENT AXBOROT TEXNOLOGIYALARI UNIVERSITETI

FARG’ONA FILIALI

“ KOMPYUTER INJINIRINGI ” FAKULTETI
DASTURIY INJINIRING KAFEDRASI
“C++ DA DASTURLASH”
FANIDAN

Kurs ishi

631-14 guruh

Bajardi: **QALANDAROV M**

Qabul qildi: **RAXMONOV S**

Farg’ona 2015

Mavzu: Injinerlik kalkulyatorini yaratish.

Reja:

1. **Kirish.**
2. **Nazariy qism.**
   - **2.1 C++ dasturlash tilining tarixi.**
   - **2.2 C++ da o‘zgaruvchilar turlari va tavsiflari**
   - **2.3 C++ tili va ob’ektlarga mo‘ljallangan dasturlash.**
3. **Asosiy qism.**
   - **3.1. Dasturni loyixalash.**
   - **3.2. Dasturning tadbiq etilishi.**
4. **Xulosa.**
5. **Foydalanilgan adabiyotlar.**
6. **Ilova**

Kirish.

Ma’lumki, insoniyat ibtidoiy jamoa tuzumi davridanoq o‘zining jismoniy mehnatini yengillatish maqsadida ko‘plab texnologiyalarni vujudga keltirgan. Ya’ni ular qo‘l mehnatini engillatish uchun qurol yasab texnik mexanizmlar yaratishga asos soldi.

Jamiyat taraqqiyotining olg‘a siljishi eng avvalo inson omiliga bog‘liqdir. Shuning uchun ham inson o‘z tafakkuri, aql-zakovatini ko‘proq ijodiy ishlarga jalb qilish va shartligi e’tirof etilmoqda. Yangidan-yangi texnik qurilma va vositalarni kashf qilish, insonning o‘z yashash sharoitiga, qilayotgan ishiga, ilmiy-texnik izlanishlariga ijodiy yondashish samarasidir. XX asrga kelib insoniyat qo‘l mehnatinigina emas, balki aqliy mehnatini ham engillatish ustida anchagina izlanishlar olib bordi. Bu yo‘lda ko‘plab texnik qurilmalar yaratildi va amaliyotga tadbiq etildi. Xuddi shunday texnik qurilmalardan biri kompyuter - elektr hisoblash mashinalari (EHM) dir.

Haqiqatan ham orzu qilish, uni ro‘yobga chiqarishga intilish insonga hos fazilat. Inson o‘z orzularini ro‘yobga chiqarishda ilm-fan erishgan yutuqlardan ham keng foydalanishga harakat qildi. Hattoki fantast yozuvchilarning bashorat qilgan yoki olimlar faraz qilgan ilmiy-texnik kashfiyotlardan ham oqilona foydalana oldi. Darxaqiqat insonning olam va odamni, tabiat va qonuniyatni bilishga bo‘lgan azaliy intilishida uni yuksak aql-zakovati bilan yaratilgan texnik qurilmalar, aqlli mashinalar kompyuterning ko‘magi beqiyosdir. Hisoblash ishlarining tarixi odamzod paydo bo‘lishidan boshlanadi.

Insoniyatning o‘z mehnatini engillatishga muttasil intilishi ham informatsiyalar rolini oshirmoqda, uni qayta ishlash, tahlil va sintez qilish jarayonlarini hal qilishni taqozo etmoqda. Dunyoning turli nuqtalarida turib o‘zaro hamkorlik ishlarini olib borish, birgalikda yangi texnika va texnologiyalar ustida ish olib borish, umuman olganda o‘zaro tez axborot almashish anchagina muammolar tug‘dirardi. XX asrda qilingan eng katta ishlardan biri bu Internet texnologiyalarining vujudga kelishi bo‘ldi. Endi Internet yordamida insoniyat

dunyoning qaysi burchagida faoliyat ko‘rsatishdan qat’iy nazar doimo bir-biri bilan aloqada bo‘lib turadi. Axborot tez almashishi o‘z navbatida inson faolligini ham oshirib yubordi va qisqa vaqtlar ichida juda ham katta yutuqlarga erishildi.

2. Nazariy qism.

Keyingi yillarda mamlakatimiz ilm-fani ham axborotlashtirishning nazariy asoslariga katta hissa qo‘shib kelmoqda, shu bilan birgalikda, hodisalar, jarayonlarni yagona axborot asosida tadqiq etishning ilmiy yo‘nalishlarini tahlil va sintez qilish natijasi bo‘lgan fan-informatikaning vujudga kelishiga boshlang‘ich nuqta qo‘yildi. Axborot, energiya, vazn, bo‘shliq va vaqtni bir butun holda batafsil o‘rganish hozirgi vaqtda inson hayotining barcha jabhalarida muhim axamiyatga ega bo‘lib qolmoqda.

Mamlakatimizda «Kadrlar tayyorlash Milliy dasturi»ning birinchi bosqichi yakunlanib, ikkinchi - sifat bosqichiga o‘tildi. O‘tgan davrda yaratilgan me’yoriy-xuquqiy hujjatlar fanlar bo‘yicha o‘quv adabiyotlarining yangi avlodini yaratishga asos bo‘lib, o‘quv jarayonini sifatini oshirishga xizmat qiladi. O‘quv adabiyotlarining yangi avlodini yaratish, ularni tayyorlash borasidagi ilmiy-uslubiy, tashkiliy va iqtisodiy masalalarni hal qilish, uzlo‘qsiz ta’lim tizimida Kadrlar tayyorlash milliy dasturi maqsadlariga erishishni ta’minlashga qaratilgan tadbirlarning ishlab chiqishni talab qiladi.

Axborot jamiyatni rivojlantiruvchi va uning tarakkiyotiga asos buluvchi muxim vosita xisoblanadi. Shu kabi axborot insoniyat tarixida eng muxim iktisodiy kursatkichlardan biri bulsa, jamiyatni kompyuterlashtirish esa iqtisodiyotni tarkibiy jixatdan qayta qurishda asosiy xarakatlantiruvchi kuchdir. Jamiyatni axborotlashtirish,yangi axborot texnologiyalari bilan ta’minlash insonlarning turli-tuman ma’lumotlarga bo‘lgan extiyojini qondirish muxim o‘rin tutadi. Inson axborot olami ichra yasharkan, voqea-xodisalar va jarayonlarning bir-biriga aloqadorligini, o‘zaro munosabati va mohiyatini taxlil etish, o‘z xayotidan kelib chiqayotgan murakkab savollarga ilmiy javob topish maksadida ko‘pdan-ko‘p dalil va raqamlarga murojaat qiladi. Axborot tufayli nazariya amaliyot bilan

birikadi. Amaliyot nazariyasiz, nazariya amaliyotsiz mavjud xam bulmaydi, rivojlanmaydi xam.

Xozirgi kunda xar bir tashkilot, o‘quv muassasasi, firma va ishlab chiqarishning barcha soxalarida raxbar va xodimlar faoliyatining samaradorligini oshirish maqsadida boshqaruv jarayonlarini ma’lum darajada avtomatlashtirishga oid muammolarni echish bilan shug‘ullanadi. Bunda ular maxsus firmalarning mutaxassislari bilan uchrashadi, ularning faoliyati bilan yaqindan tanishadi, ular ishlab chiqaradigan maxsulotlarni ko‘radi va pirovardida o‘zida avtomatlashtirish uchun kerak bo‘ladigan texnik jixozlarni xarid qiladi. Albatta, tashkilotlarga o‘rnatilgan avtomatlashtirish jixozlari yildan-yilga yangilanib, texnik jixatdan takomillashtirib boriladi.

XX asrning so‘nggi o‘n yili mobaynida axborotlar bilan ishlash va axborotlashtirish juda rivojlandi. Bunga sabab shundaki, kundalik turmushda axborotlar, ularni qayta ishlash va uzatishning axamiyati ortib bormoqda. Bu esa o‘uz navbatida jamiyatning xar bir a’zosidan axborotlashtirish va axborot texnologiyalari sirlarini, uning qoida va qonuniyatlarini mukammal bilishni takozo etadi.

2.1. C++ dasturlash tilining tarixi

Birinchi elektron hisoblash mashinalari paydo bo‘lishi bilan dasturlash tillari evolyusiyasi boshlanadi. Dastlabki kompyuterlar ikkinchi jahon urushi vaqtida artilleriya snaryadlarining harakat traektoriyasini hisob-kitob qilish maqsadida qurilgan edi. Oldin dasturchilar eng sodda mashina tilini o‘zida ifodalovchi kompyuter komandalari bilan ishlaganlar. Bu komandalar nol va birlardan tashkil topgan uzun qatorlardan iborat bo‘lar edi. Keyinchalik, insonlar uchun tushunarli bo‘lgan mashina komandalarini o‘zida saqlovchi (masalan, ADD va MOV komandalari) assembler tili yaratildi.

Bu komandalarni mashina tiliga interpretatorlar va kompilyatorlar ko‘chirar edi. Interpretator dasturni o‘qish jarayonida uning komandalarini ketma - ket mashina tiliga o‘tkazadi. Kompilyator esa yaxlit programma kodini biror bir oraliq forma - oboekt fayliga o‘tkazadi. Bu bosqich kompilyasiya bosqichi deyiladi.

Bundan so‘ng kompilyator ob’ektli faylni bajariluvchi faylga aylantiradigan kompanovka dasturini chaqiradi. Interpretatorlar bilan ishlash osonroq, chunki dastur komandalari qanday ketma - ketlikda yozilgan bo‘lsa shu tarzda bajariladi. Bu esa dastur bajarilishini nazorat qilishni osonlashtiradi. Kompilyator esa kompilyasiya va kompanovka kabi qo‘shimcha bosqichlardan iborat bo‘lganligi uchun ulardan hosil bo‘ladigan bajariluvchi faylni tahlil qilish va o‘zgartirish imkoniyati mavjud emas. Faqatgina kompilyasiya qilingan fayl tezroq bajariladi, chunki bundagi komandalar kompilyasiya jarayonida mashina tiliga o‘tkazilgan bo‘ladi.

C++ kabi kompilyasiya qiluvchi dasturlash tillarini yana bir afzalligi hosil bo‘lgan dastur kompyuterda kompilyatorsiz ham bajarilaveradi. Interpretatsiya qiluvchi tillarda esa tayyor dasturni ishlatish uchun albatta mos interpretator dasturi talab qilinadi

2.2. 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.

Apparat taminoti 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 bir o‘lchamiga ega. Butun son ko‘rsatkichni saqlash uchun etarli 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. 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.

2.3. C++ tili va ob’ektlarga mo‘ljallangan dasturlash.

Inkapsulyatsiyalash

Inkapsulyasiyalash 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 inkapsulyasiyalash tufayli mustaqillik darajasi ortadi, chunki ichki detallar interfeys ortida yashiringan bo‘ladi. Inkapsulyasiyalash modullikning ob’ektga mo‘ljallangan tavsifidir. Inkapsulyasiyalash 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 inkapsulyasiyalash atamasi «germetik berkitilgan; tashqi ta’sirlardan himoyalangan dastur qismi» degan ma’noni bildiradi. Agar biron-bir dasturiy ob’ektga inkapsulyasiyalash qo‘llangan bo‘lsa, u holda bu ob’ekt qora quti sifatida olib qaraladi. Siz qora quti nima qilayotganini uning tashqi interfeysini ko‘rib turganingiz uchungina bilishingiz mumkin. Qora quti 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 ob’ektlar ushbu ob’ektga qanday talablar yuborishi mumkinligi ko‘rsatilgan bo‘ladi. Interfeys – ob’ektni boshqarish pulti.

Ommaviy, xususiy va himoyalangan kirishlar.

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 hil funksiyalarni bajaradi. Ob’ektga mo‘ljallangan tillarning ko‘pchiligida kirishning uchta darajasi mavjud.

- Ommaviy (public) – barcha ob’ektlar uchun kirish uchun ruxsat bor;

- Himoyalangan (protected) - faqat ushbu ekzemplyarga va har qanday tarmoq sinflarga kirishga ruxsat bor;

- Xususiy (private) - faqat ushbu ekzemplyarga kirishga ruxsat bor.

Loyihada kirish darajasini to‘g‘ri tanlab olish g‘oyat muhimdir. 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.

Inkapsulyatsiyalash nima uchun kerak?

Inkapsulyasiyalashdan to‘g‘ri foydalanish tufayli ob’ektlar bilan o‘zgartiriladigan komponentlar (tarkibiy qismlar) dek muomala qilish mumkin. Boshqa ob’ekt sizning ob’ektingizdan foydalana olishi uchun, u sizning ob’ektingizning ommaviy interfeysidan qanday foydalanish kerakligini bilishi kifoya. Bunday mustaqillik uchta muhim afzallikka ega.

Mustaqillik tufayli, ob’ektdan takroran foydalanish mumkin. Inkapsulyasiyalash puxta amalga oshirilgan bo‘lsa, ob’ektlar ma’lum bir programmaga bog‘lanib qolgan bo‘lmaydi. Ulardan imkoni bo‘lgan xamma erda foydalanish mumkin bo‘ladi. Ob’ektdan boshqa biron o‘rinda foydalanish uchun, uning interfeysidan foydalanib qo‘ya qolish kifoya.

Inkapsulyasiyalash tufayli, ob’ektda boshqa ob’ektlar uchun ko‘rinmas bo‘lgan o‘zgarishlarni amalga oshirish mumkin. Agar interfeys o‘zgartirilmasa, barcha o‘zgarishlar ob’ektdan foydalanayotganlar uchun ko‘rinmas bo‘ladi. Inkapsulyasiyalash komponentni yaxshilash, amalga oshirish samaradorligini ta’minlash, hatolarni bartaraf etish imkonini beradi, yana bularning hammasi dasturning boshqa ob’ektlariga ta’sir ko‘rsatmaydi. Ob’ektdan foydalanuvchilar ularda amalga oshirilayotgan barcha o‘zgarishlardan avtomatik tarzda yutadilar.

Himoyalangan ob’ektdan foydalanishda ob’ekt va dasturning boshqa qismi o‘rtasida biron-bir ko‘zda tutilmagan o‘zaro aloqalar bo‘lishi mumkin emas. Agar ob’ekt boshqalardan ajratilgan bo‘lsa, bu holda u dasturning boshqa qismi bilan faqat o‘z interfeysi orqali aloqaga kirishishi mumkin. Shunday qilib, inkapsulyasiyalash yordamida modulli dasturlarni yaratish mumkin. Samarali inkapsulyasiyalashning uchta o‘ziga xos belgisi qo‘yidagicha:

-abstraksiya;

-joriy qilishning berkitilganligi;

-mas’uliyatning bo‘linganligi.

Abstraksiya

Garchi ob’ektga mo‘ljallanganliklar inkapsulyasiyalashdan foydalanishga yordam bersada, biroq ular inkapsulyasiyalashni kafolatlamaydi. Tobe va ishonchsiz kodni yaratib qo‘yish oson. Samarali inkapsulyasiyalash – sinchkovlik bilan ishlab chiqish xamda abstraksiya va tajribadan foydalanish natijasidir. Inkapsulyasiyalashdan samarali foydalanish uchun dasturni ishlab chiqishda avval abstraksiyadan va uning bilan bog‘liq konsepsiyalardan foydalanishni o‘rganib olish lozim.

Abstraksiya murakkab masalani soddalashtirish jarayonidir. Muayyan masalani echishga kirishar ekansiz, siz barcha detallarni hisobga olishga o‘rinmaysiz, balki echimni 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 hasharotlar va qushlar xam 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 echimini 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 biron-bir ma’lum masala echimiga 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 echishga qaratilgan ob’ektlarni yaratishga harakat qiling. Abstraksiya bitta masala echimidan ushbu sohadagi boshqa masalalarni ham echishda foydalanish imkonini beradi.

Ikkita misolni ko‘rib chiqamiz.

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‘rsatildi» 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 erda boshqalaridan ko‘proq vaqt turib qolgan bo‘ladi.

Restoranlar «birinchi kelganga birinchi bo‘lib hizmat ko‘rsatildi» algoritmi bo‘yicha ishlaydi.

Garchi bu misollar butkul turlicha bo‘lsada, 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 hizmat ko‘rsatildi» algoritmi qo‘llangan. Bu o‘rinda navbat elementi nimani bildirishi muhim emas. Haqiqatda ushbu element navbat oxiriga kelib qo‘shilishi hamda navbatni uning boshiga etganda 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 hizmat ko‘rsatildi» algoritmi bo‘yicha ishlov beriladi. Samarali abstraksiyani bajarish uchun bir nechta qoidalarni ifodalash mumkin.

-Qandaydir aniq xolatni emas, umumiy holatni olib qarang.

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

-Garchi abstraksiya g‘oyat qimmatli bo‘lsada, biroq eng echimli masalani yodingizdan chiqarmang.

-Abstraksiya hamma vaqt xam ochiq-oydin emas. Masalani echar ekansiz, siz birinchi, ikkinchi va, hatto, uchinchi martasiga 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. Yaxshigina evristik qoida mavjud, bo‘lib, unga ko‘ra, agar siz biron bir masalani o‘zaro o‘hshash usullar bilan kamida uch marta echgan bo‘lsangiz, abstraksiyani faqat shunday masalalarga qo‘llash tavsiya qilinadi.

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

Joriy qilishni yashirish yordamida sirlarni yashirish.

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

Joriy qilishning berkitilganligi

Joriy qilishning berkitilganligi ikkita afzallikka ega:

- ob’ektlarni foydalanuvchilardan himoyalaydi;

- foydalanuvchilarni ob’ektlardan himoyalaydi.

Birinchi afzallik - ob’ektlarni himoyalashni ko‘rib chiqamiz.

Asl inkapsulyasiyalash 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 inkapsulyasiyalashni ko‘llashni osonlashtiradi, chunki ular tufayli inkapsulyasiyalashni vorisliksiz va polimorfizmsiz qo‘llash mumkin, bu esa inkapsulyasiyalashning 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 ob’ektga muljallangan yondoshuv 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 hotirasida bitlar ko‘rinishida namoyon bo‘lsada, siz bitlar ustidagi operatsiyalar haqida bosh qotirib o‘tirmaysiz.

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

Joriy qilishning berkitilganligi dasturni moslashuvchan qiladi, chunki foydalanuvchilar ob’ektning joriy qilinishini hisobga olishga majbur emaslar. Shunday qilib, joriy qilishning berkitilganligi nafaqat ob’ektni ximoyalaydi, balki kuchsiz bog‘langan kodni yaratishga yordam berib, ushbu ob’ektdan 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.

Inkapsulyasiyalash 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 inkapsulyasiyalashning afzalliklari yo‘qoladi: mustaqil va takroran qo‘llanadigan ob’ektlarning 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 bajarayapsiz. Interfeysni belgilashda, uni taqdim etishgina emas, balki joriy qilishda qo‘llangan turlarning o‘ziga hos tomonlarini hujjatlashtirish ham g‘oyat muhimdir. Biroq, ommaviy interfeysning har qanday boshqa qismi kabi, hulq-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 xam 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 ma’suliyat tushunchasi bilan bog‘liqligi tabiiydir. Kuchsiz bog‘langan dasturni yaratish uchun, ma’suliyatni tegishli ravishda taqsimlash ham muhimdir. Ma’suliyat tegishli ravishda taqsimlanganda,

har bir ob’ekt o‘zi mas’ul bo‘lgan bitta funksiyani bajaradi hamda bu funksiyani yaxshi bajaradi. Bu esa ob’ekt bir butunlikni tashkil etishini ham bildiradi. Boshqacha qilib aytganda, funksiyalar va o‘zgaruvchilarning tasodifiy to‘plamiga ehtiyoj bo‘lmaydi. Inkapsulalanayotgan ob’ektlar o‘rtasida yaqin konseptual aloqa bo‘lmog‘i kerak. Barcha funksiyalar umumiy vazifani bajarmog‘i kerak.

Joriy qilish berkitilmas ekan, mas’uliyat ob’ektdan chetga chiqib ketishi mumkin. Biroq o‘z vazifasini qanday hal qilishni aynan ob’ektning o‘zi bilishi lozim, ya’ni aynan ob’ekt 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 ob’ekt bir hil 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 ob’ektga bitta (har holda kam miqdordagi) vazifa uchun mas’uliyatni yuklash lozim. Agar bitta ob’ektga ko‘p miqdordagi vazifalar ustidan mas’uliyat yuklatib qo‘yilgan bo‘lsa, ularni bajarish murakkablashadi, ob’ektni kuzatib borish va takomillashtirish ham qiyinlashadi. Mas’uliyatni o‘zgartirish ham havfli, chunki bunda, agar ob’ekt bir necha xulq-atvor liniyalariga ega bo‘lsa, ularni ham o‘zgartirishga to‘g‘ri keladi. Natijada juda katta miqdordagi axborot bir erga jamlanib qoladi, uni esa teng taqsimlash lozim. Ob’ekt 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 inkapsulyasiyalash umuman qo‘llanmagan dasturda yuzaga keladigan barcha muammolarga duch kelib qolasiz.

Ob’ekt bir-ikkitadan ortiq vazifa uchun mas’ul ekanini aniqlagach, mas’uliyatning bir qismini boshqa ob’ektga olib o‘tish kerak.

Joriy etishning berkitilganligi - samarali inkapsulyasiyalash yo‘lidagi qadamlardan biri xolos. Mas’uliyatni tegishli ravishda taqsimlamay, natijada siz protseduralar ro‘yxatiga ega bo‘lib qolasiz xolos.

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.

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 havfli bo‘lishi mumkin. Hatto agar siz biron-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 echish kerak.

Sinfga masalani echish uchun kerak bo‘lganidan ko‘proq narsani kiritish tavsiya qilinmaydi. Birdaniga barcha masalalarni echmang, e’tiboringizni bittasining echimiga qarating. Va shundan so‘nggina qilib bo‘lingan ishga nisbatan abstraksiyani qo‘llash usulini izlab ko‘rish mumkin.

Masalan, katta hisoblar yoki murakkab modelga o‘hshash ancha murakkab masalalar ham uchraydi. Bu o‘rinda gap mas’uliyatni taqsimlash nuqtai nazaridan murakkablik haqida bormoqda. Ob’ektning 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 abstaksiyalashdan foydalanishga o‘rganish uchun vaqt kerak. Haqiqiy abstrakt dastur haqiqiy hayot talablariga asoslangan bo‘lmog‘i lozim. U dasturchi shunchaki takroran qo‘llanadigan ob’ektni yaratishga jazm

qilganligi natijasida yuzaga kelmaydi. Aytganlaridek, ihtiroga extiyoj tug‘ilganidagina, u tug‘iladi. Xuddi shu tamoyil ob’ektlarni yaratishda ham amal qiladi. Birinchi martadayoq haqiqatan abstrakt, takroran qo‘llanadigan ob’ektni yozish mumkin emas. Odatda takroran qo‘llanadigan ob’ektlar ishda sinovdan o‘tib bo‘lgan hamda ko‘plab o‘zgarishlarga uchragan dasturni takomillashtirish jarayonida yaratiladi.

Ichki o‘zgaruvchilarni hamma vaqt berkitish kerak: ular konstantalar bo‘lgan holatlar bundan mustasno. Muhimi, ular nafaqat berkitilgan 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.

Inkapsulyasiyalashning asosiy afzalliklari

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

Kodning muhim uchastkalarini 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 xolda funksional tavsiflarni takomillashtirish mumkin. Buning ustiga, joriy qilish berkitilgan ekan, takomillashtirilgan komponentdan foydalanayotgan kodning ishga tushirilish tavsiflari avtomatik tarzda yahshilanadi: 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. Ob’ektning o‘zgarishi ushbu ob’ekt 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 «nul»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 hususiyatlari, hulq-atvori va joriy qilinishiga vorislik qiladi. Avval mavjud bo‘lgan sinf interfeysining barcha metodlari va hususiyatlari avtomatik tarzda voris interfeysida paydo bo‘ladi. Vorislik voris sinfida biron-bir jihatdan to‘g‘ri kelmagan hulq-atvorni avvaldan ko‘ra bilish imkonini beradi. Bunday foydali hususiyat dasturiy ta’minotni talablarning

o‘zgarishiga moslashtirish imkonini beradi. Agar o‘zgartirishlar kiritishga ehtiyoj tug‘ilsa, bu xolda 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, ob’ekt ishini o‘zgartirish imkonini beradi. Axir bu holda qayta test sinovlaridan puxta o‘tkazilgan asosiy sinflarga tegmasa ham bo‘ladi.

Agar siz ko‘p martalab qo‘llash yoki boshqa biron maqsadlarga ko‘ra vorislikni qo‘llashga ahd qilsangiz, avval har gal qarang - vorislik-sinf bilan vorislikni berayotgan sinfning turlari o‘zaro mos keladimi. Vorislikda turlarning mos kelishi ko‘pincha «Is-a» testi deb ataladi. Ikkita sinf bir hil 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 «Xas-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 xas a tail). Shuning uchun Tail sinfining ekzemplyari (nushasi)ni, masalan, Canine ga kiritib qo‘yish kerak. Ushbu biroz biologik misoldan ko‘rinib turganidek, «Ia-a» munosabati sinflarning tabaqalanishida namoyon bo‘ladi. Sinflar orasidagi «Xas-a» munosabati esa shunga olib keladiki, bir sinf ikkinchisida mavjud bo‘ladi.

Boshqa sinfga vorislik bo‘layotgan sinf voris 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. Vorislik sinf o‘z ajdodi bo‘lgan sinfdan hususiyatlar va hulq-atvorni voris qilib olayotganida, u shuningdek o‘z ajdodi bo‘lgan sinf ehtimol boshqa sinflardan voris 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. Vorislik sinfga funksiyalarni kengaytirish va yangilarini qo‘shish uchun ruxsat beriladi. Ammo unga funksiyalarni chiqarib tashlashga ruxsat yo‘q.

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

- O‘rniga o‘rin qo‘yish (almashtirish): yangi sinf ajdodlarining metodi yoki hususiyatini shunchaki o‘zlashtirib olmaydi, balki unga yangi ta’rif ham beradi;

- YAngi: yangi sinf butunlay yangi metodlar yoki hususiyatlarni qo‘shadi;

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

Ob’ektga mo‘ljallangantillarning ko‘pchiligi ta’rifni ma’lumot o‘zatilgan ob’ektdan qidiradilar. Agar u erdan 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 ko‘rsatadi.

Voris sinflar himoyalangan kirish darajasiga ega bo‘lgan metodlar va hususiyatlarga kirish xuquqini 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 yondoshuv 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 voris qilib olingan realizatsiya (joriy qilinish) bilan bog‘liq bo‘lasiz. Vorislikning bu turini ehtiyotkorlik bilan qo‘llash lozim. Yaxshisi bu o‘rinda «Xas-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 hatolar miqdorini ham kamaytiradi.

Almashtirish imkoniyati - OMYO da muhim tushunchalardan biri. Vorislik 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 vorislik sinfni yaratishda hulq-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 ob’ektlardan qanday foydalanishni biladi.

Polimorfizm

Agar inkapsulyasiyalash va vorislikni OMYO ning foydali vositalari sifatida olib qarash mumkin bo‘lsa, polimorfizm - eng universal va radikal vositadir.

Polimorfizm inkapsulyasiyalash va vorislik bilan chambarchas bog‘liq, boz ustiga, polimorfizmsiz OMYO samarali bo‘lolmaydi. Polimorfizm - OMYO paradigmasida markaziy tushunchadir. Polimorfizmni egallamay turib, OMYO 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 hulq-atvorni bildirishi mumkin.

Vorislik polimorfizmning ayrim turlaridan foydalanish uchun zarurdir. Aynan o‘rindoshlik imkoniyati mavjud bo‘lgani uchun, polimorfizmdan foydalanish mumkin bo‘ladi. Polimorfizm yordamida tizimga to‘g‘ri kelgan paytda qo‘shimcha funksiyalarni qo‘shish mumkin. Dasturni yozish paytida hatto tahmin 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, uning tufayli tarmoq sinf nushalari o‘zini turlicha tutishi mumkin. Qo‘shilish polimorfizmidan foydalanib, yangi tarmoq sinflarni kiritgan holda, tizimning hulq-atvorini o‘zgartirish mumkin. Uning bosh afzalligi shundaki, dastlabki dasturni o‘zgartirmay turib, yangi hulq-atvorni yaratish mumkin.

Aynan polimorfizm tufayli joriy qilishdan takroran foydalanishni vorislik bilan aynanlashtirish kerak emas. Buning o‘rniga vorislikdan avvalam bor o‘zaro almashinish munosabatlari yordamida polimorfizm hulq-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 ob’ektni 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 inkapsulyasiyalash va vorislikdan samarali foydalanishdir. Inkapsulyasiyalashsiz 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. Hamma vaqt bazaviy sinfga imkon darajada yaqinlashtirilgan darajada dasturlashga o‘ringan holda, o‘rinbosarlik munosabatlarini o‘rnatishga harakat qilish kerak. Bunday usul dasturda ishlov berilayotgan ob’ektlar turlari miqdorini oshiradi.

Puxta o‘ylab ishlab chiqilgan tabaqalanish o‘rinbosarlik munosabatlarini o‘rnatishga yordam beradi. Umumiy qismlarni abstrakt sinflarga olib chiqish kerak hamda ob’ektlarni shunday dasturlash kerakki, bunda ob’ektlarning ixtisoslashtirilgan nushalari 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.

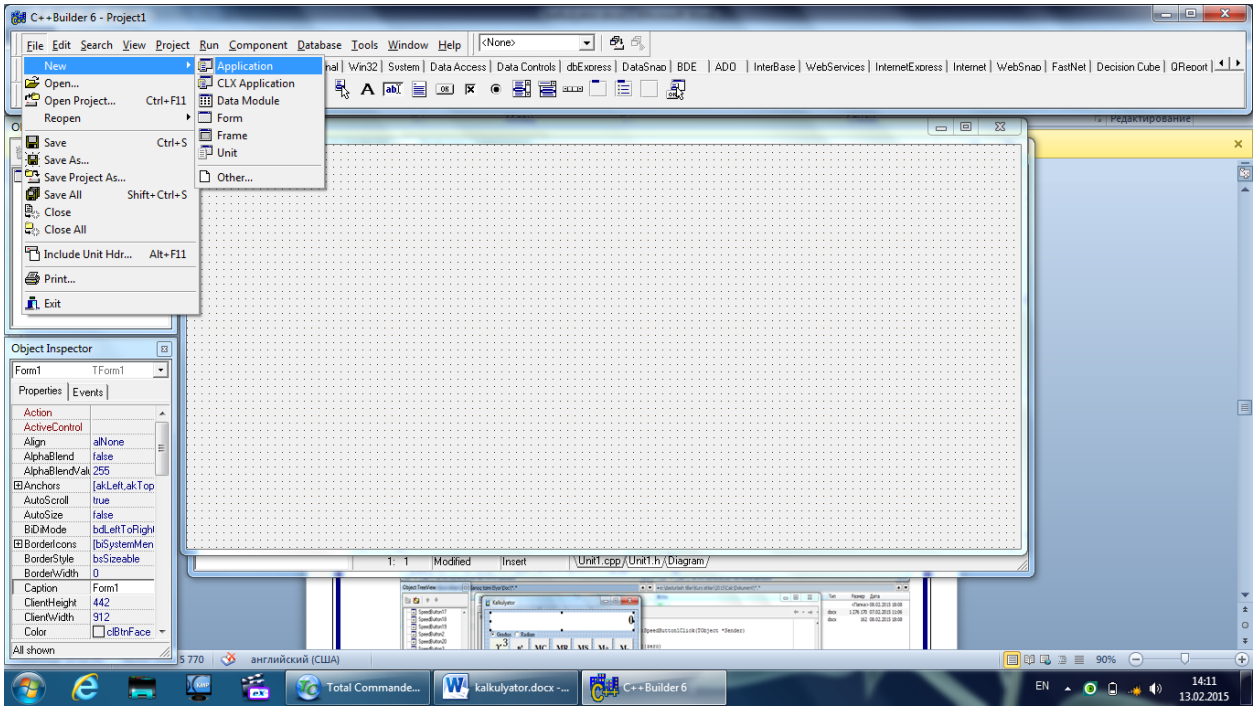
Biroq ko‘p o‘rinlarda tajribasiz loyihachilar polimorfizmni kuchaytirish maqsadida xulq-atvorni juda baland tabaqaviy darajaga olib chiqishga urinadilar. Bu holda har qanday avlod ham bu hulq-atvorni ushlab tura oladi. Shuni esdan chiqarmaslik kerakki, avlodlar o‘z ajdodlarining funksiyalarini chiqarib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

3. Asosiy qism.

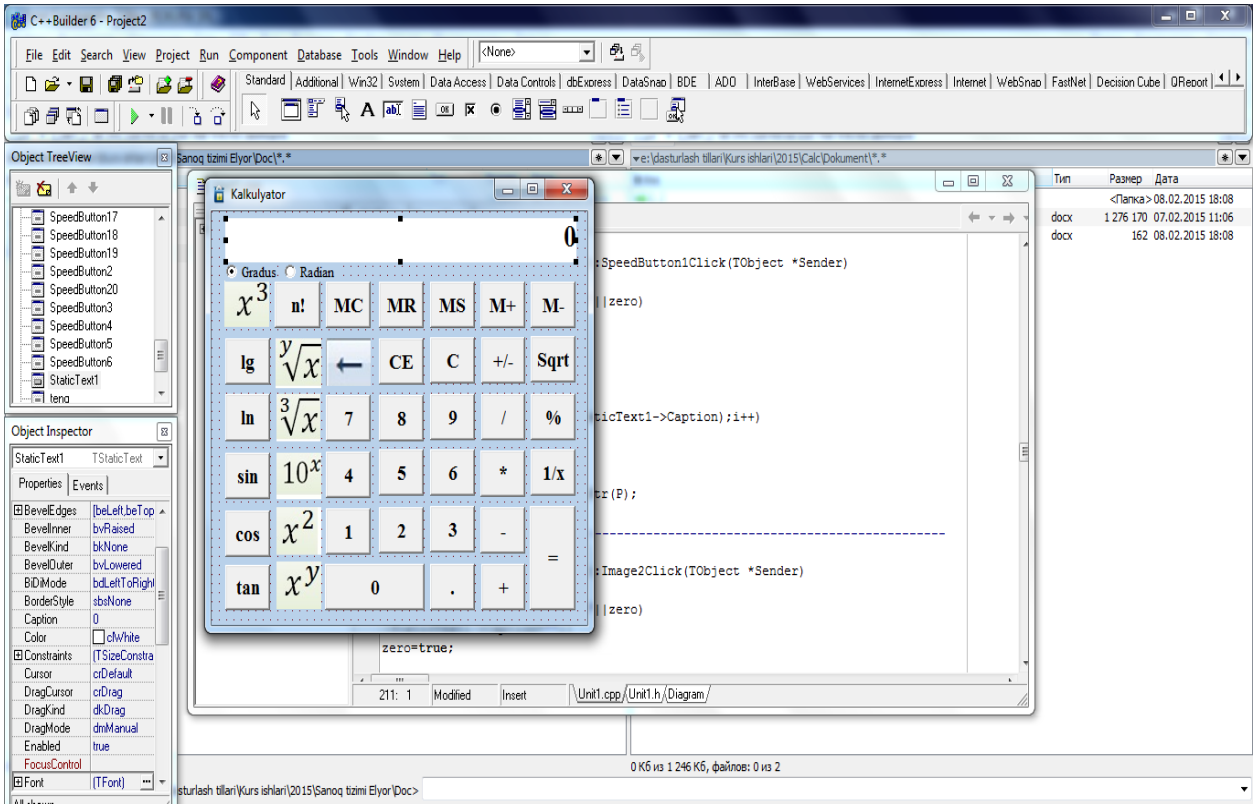
3.1. **Dasturni loyixalash.**

Dasturni tuzishda uning interfeysini qanday yaratish to’g’risida ko’p o’yladim va foydalanuvchilarga qulayroq bo’lishi uchun Microsoft Windows operatsion tizimi standart dasturi Kalkulyator dasturining Injiner interfeysiga o’xshatishga qaror qildim. Chunki bu interfeys foydalanuvchilarga qulay bo’lishi bilan bir qatorda, yetuk dasturchilar tomonidan tuzilgan hisoblanadi.

Dastur interfeysini C++ dasturlash tili asosida Borland C++ Builder 6 dasturida tashkil etdim. Dastlab C++ Builder 6 dasturini ishga tushirdim. File menyusidan New→ Application tanlaymiz.

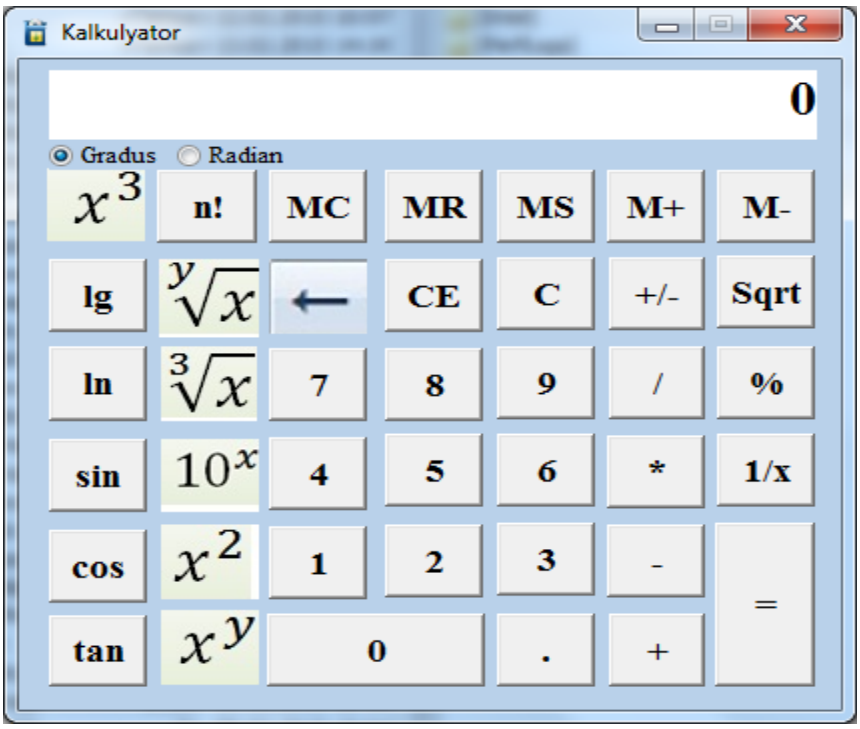


Dastur interfeysini tuzishda StaticText, SpitButton, RadioButton, ildiz, darajalarni xisoblash uchun Image komponentalaridan foydalandim.

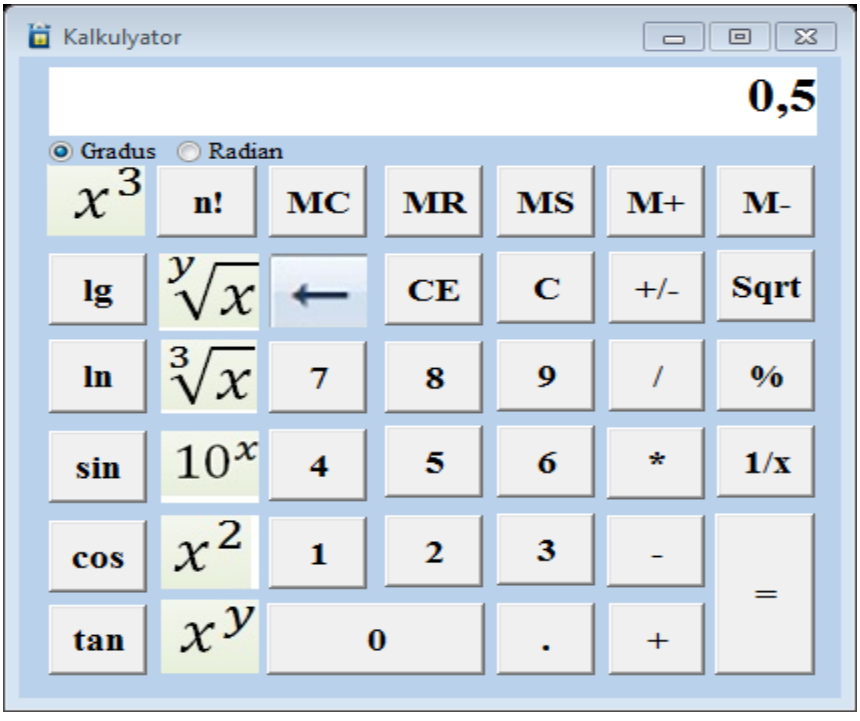


Injinerlik kalkulyator dasturiy ta’minotini yaratish uchun men matematika fani bo’yicha bilimlarimni takrorlab oldim. Trigonometrik funksiyalar, logarifmlar, faktorial, ildiz, darajaga ko’tarish kabi bir qancha matematik ifoda va funksiyalarni takrorlab, bu ifoda va funksiyalar C++ dasturlash tilida qanday tartibda yozilish sintaksisini o’rganib chiqdim.

3.2. Dastur ishlatilganda asosiy forma hosil bo’ladi. Men asosiy formani quyidagicha dizaynda hosil qildim.

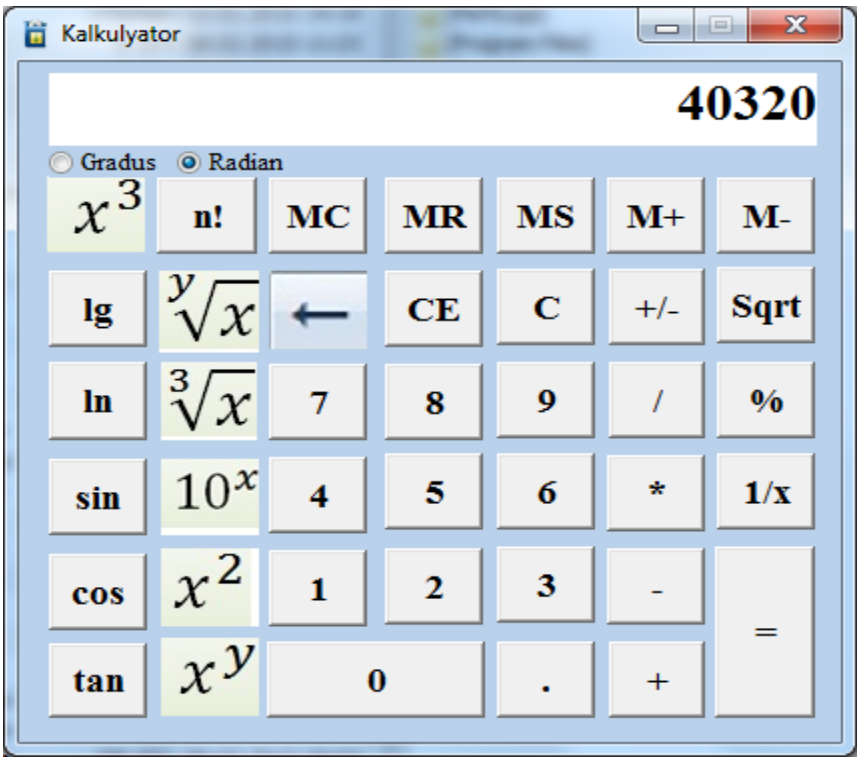


Kalkulyator dasturidan foydalanish xuddi Windows standart kalkulyatori singari amalga oshiriladi. Trigonometrik funksiyalarni xisoblashda qulaylik maqsadida gradus va radian xisoblashni tashkil qildim. Masalan 30^0 da sinusning qiymati:

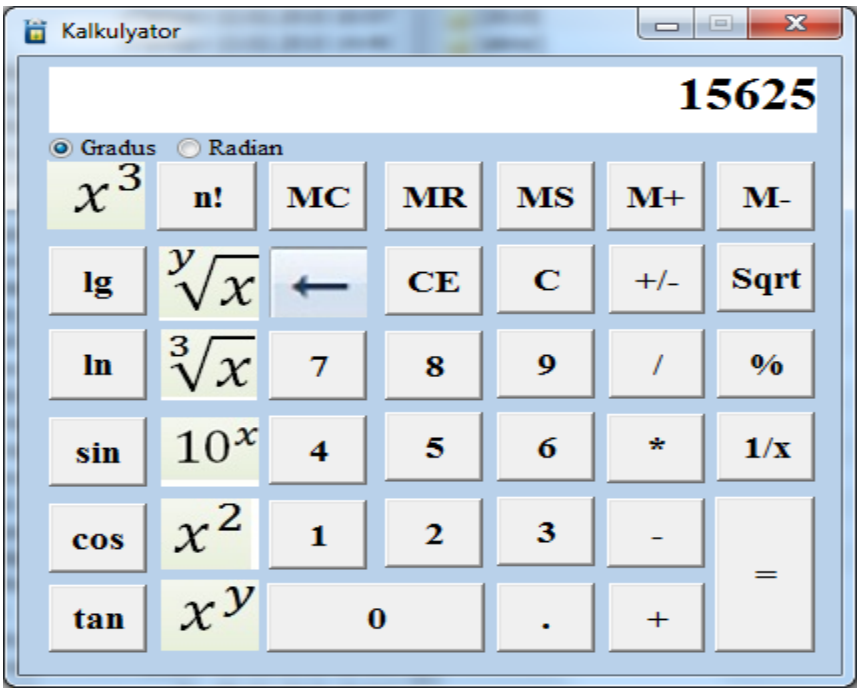


Boshqa imkoniyatlar:

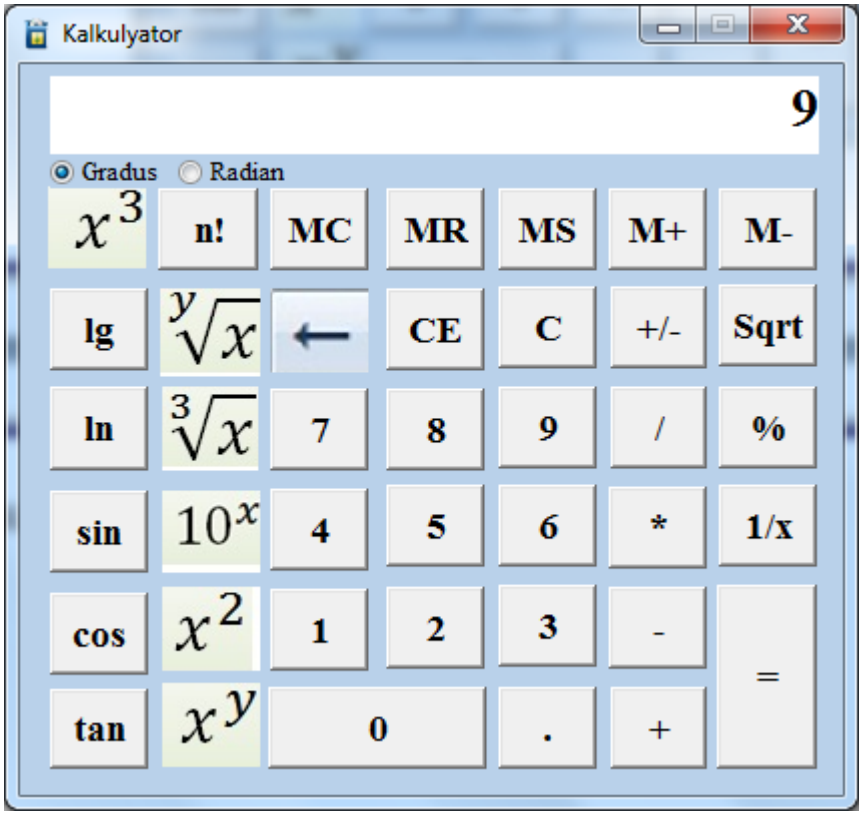
Masalan: 8! faktarialni xisoblash:



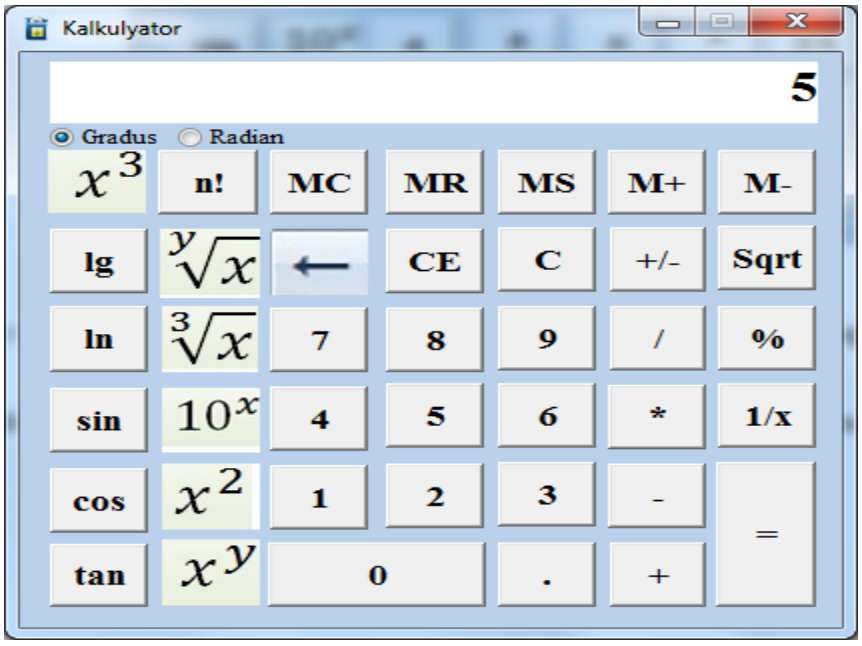
x^y ni xisoblash: birinchi x ning qiymati kiritiladi x^y tugmasi bosiladi so’ng y ning qitmati kiritilib $=$ tugmasi bosiladi. Masalan 5 ning 6- darajasi:



Kvadrat ildizdan foydalanish: Masalan 81 ning kvadrat ildizi:



x dan y dara jali ildiz olish: birinchi x ning qitmati kiritilib $\sqrt[y]{x}$ tugmasi bosiladi so’ng y ning qiymati kiritilib = tugmasi bosiladi. Masalan 125 ning 3 ildizi:



4. Xulosa.

Hammamizga ma’lumki, XXI asr “Axborot texnologiyalari asri” hisoblanadi. Bu asrda dasturlash sohasida misli ko‘rilmagan o‘zgarishlar bo‘ldi. Bunda avtomatlashtirish dasturning asosini tashkil qiladi. Yuqorida yaratilgan dastur ham foydalanuvchiga turli qulayliklar yaratib, hisob ishlaridan ozod qiladi. Men dastur tuzish davomida C++Builderda keng imkoniyatlar yaratilganligini tushundim.

Men ushbu kurs ishini bajarish davomida C++ dasturlash tilida dasturlash hamda avtomatlashtirish to’g’risidagi bilimlarimni mustaxkamlab oldim. Dastur tuzish davomida C++ Builder da ishladim. C++ Builderda ko’plab komponentalarni ishlatdim. Kurs ishini bajarish davomida dasturdagi ko’plab komponentalar palitrasidan foydalanishni, dastur tuzishdan oldin uning algoritmini to’liq tuzib chiqish lozimligini o’rgandim. Chunki dasturchi dasturni tuzishdan oldin uning algoritmini ishlab chiqib dasturni qanday natija qaytarishini bilgan holdagina dastur bexato, samarali ishlar ekan. Men ushbu kurs ishini bajarish davomida matematika fani bo’yicha ham bilimlarimni mustahkamlab oldim.

5. Foydalanilgan adabiyotlar.

1. Гради Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. Невский диалект. 2012.
2. И. Грехем. Объектно-ориентированные методы. Принципы и практика. Вильямс. 2004.
3. Г.С.Иванова. Объектно-ориентированное программирование. Учебник. МГТУ им. Баумана. 2003
4. Т. Фейсон. Объектно ориентированное программирование на C++ 4.5. Киев. Диалектика. 1996.
5. Algebra. Darslik 9-sinf.2010.
6. Geometriya. Darslik 9-sinf.2010.
7. www.ziyonet.uz
8. www.google.com
9. www.daryo.uz
10. www.kutubxona.uz
11. www.dasturchi.uz
12. www.dastur.uz

6. Ilova

Dastur kodlaridan misollar.

```
#include <vcl.h>

#pragma hdrstop

#include <math.h>

#include <stdio.h>

#include "Unit1.h"

//---------------------------------------------------------------------------

#pragma package(smart_init)

#pragma resource "*.dfm"

TKalkulyator *Kalkulyator;

int amal;

double i,p,r,yig;

bool zero, yigin;

//---------------------------------------------------------------------------

__fastcall TKalkulyator::TKalkulyator(TComponent* Owner)

        : TForm(Owner)

{

}

//---------------------------------------------------------------------------

void __fastcall TKalkulyator::raqam(TObject *Sender)

{
```

```
if (StaticText1->Caption=="0"||zero)
StaticText1->Caption="";
if (zero) zero=false;
StaticText1->Caption=StaticText1->Caption+((TButton*)Sender)->Caption;
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::plusClick(TObject *Sender)
{
amal=1;
i=StrToFloat(StaticText1->Caption);
zero=true;
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::minusClick(TObject *Sender)
{
amal=2;
i=StrToFloat(StaticText1->Caption);
zero=true;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TKalkulyator::kopClick(TObject *Sender)
{
amal=3;
i=StrToFloat(StaticText1->Caption);
zero=true;
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::bulClick(TObject *Sender)
{
amal=4;
i=StrToFloat(StaticText1->Caption);
zero=true;
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::tengClick(TObject *Sender)
{
switch (amal)
{
case 1:
p=StrToFloat(StaticText1->Caption);
r=i+p;
```

```
StaticText1->Caption=FloatToStr(r);
zero=true;
break;

case 2:
p=StrToFloat(StaticText1->Caption);
r=i-p;
StaticText1->Caption=FloatToStr(r);
zero=true;
break;

case 3:
p=StrToFloat(StaticText1->Caption);
r=p*i;
StaticText1->Caption=FloatToStr(r);
zero=true;
break;

case 4:
p=StrToFloat(StaticText1->Caption);
r=i/p;
```

```
StaticText1->Caption=FloatToStr(r);
zero=true;
break;

case 5:
p=StrToFloat(StaticText1->Caption);
r=pow(i,1/p);
StaticText1->Caption=FloatToStr(r);
zero=true;
break;

case 6:
p=StrToFloat(StaticText1->Caption);
r=pow(i,p);
StaticText1->Caption=FloatToStr(r);
zero=true;
break;
}
}
//---------------------------------------------------------------------
void __fastcall TKalkulyator::bsClick(TObject *Sender)
```

```
{
if (StaticText1->Caption=="0") return;
StaticText1->Caption=StaticText1->Caption.Delete(StaticText1-
>Caption.Length(),1);
if (StaticText1==""||StaticText1=="-")
StaticText1->Caption="0";
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::ishoraClick(TObject *Sender)
{
double s=StrToFloat(StaticText1->Caption)*-1;
StaticText1->Caption=FloatToStr(s);
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::ildizClick(TObject *Sender)
{
double q=StrToFloat(StaticText1->Caption);
double ildiz=sqrt(q);
StaticText1->Caption=FloatToStr(ildiz);
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::foizClick(TObject *Sender)
```

```
{ double fo=StrToFloat(StaticText1->Caption);
  double fo2=i*fo/100.0;
  StaticText1->Caption=FloatToStr(fo2);
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::btxClick(TObject *Sender)
{
double bx=1/StrToFloat(StaticText1->Caption);
StaticText1->Caption=FloatToStr(bx);
zero=true;
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::ptozaClick(TObject *Sender)
{
StaticText1->Caption="0";
i=0;
p=0;
amal=0;
zero=false;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TKalkulyator::tozalaClick(TObject *Sender)
{
  StaticText1->Caption="0";
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::vergulClick(TObject *Sender)
{
if (!StaticText1->Caption.Pos(","))
StaticText1->Caption=StaticText1->Caption+",";
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::Image1Click(TObject *Sender)
{
if (StaticText1->Caption=="0") return;
StaticText1->Caption=StaticText1->Caption.Delete(StaticText1-
>Caption.Length(),1);
if (StaticText1==""||StaticText1=="-")
StaticText1->Caption="0";
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::SpeedButton1Click(TObject *Sender)
```

```
{
if (StaticText1->Caption=="0"||zero)
{StaticText1->Caption="";}
zero=true;
float P=1;
for(int i=1;i<=StrToFloat(StaticText1->Caption);i++)
{
 P=P*i;
}
StaticText1->Caption=FloatToStr(P);
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::Image2Click(TObject *Sender)
{
if (StaticText1->Caption=="0"||zero)
{StaticText1->Caption="";
}
zero=true;
amal=5;
i=StrToFloat(StaticText1->Caption);
zero=true;
```

```
}
//---------------------------------------------------------------------------


void __fastcall TKalkulyator::Image3Click(TObject *Sender)
{
if (StaticText1->Caption=="0"||zero)
{StaticText1->Caption="";
}
zero=true;
float son,ss, dar=3;
son=StrToFloat(StaticText1->Caption);
ss=pow(son,1/dar);
StaticText1->Caption=FloatToStr(ss);
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::Image7Click(TObject *Sender)
{
if (StaticText1->Caption=="0"||zero)
{StaticText1->Caption="";}
zero=true;
float daraja=StrToFloat(StaticText1->Caption);
```

```
StaticText1->Caption=FloatToStr(pow(10,daraja));
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::Image4Click(TObject *Sender)
{
if (StaticText1->Caption=="0"||zero)
{StaticText1->Caption="";}
zero=true;
StaticText1->Caption=FloatToStr(pow(StrToFloat(StaticText1->Caption),2));
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::Image5Click(TObject *Sender)
{
if (StaticText1->Caption=="0"||zero)
{StaticText1->Caption="";
}
zero=true;
amal=6;
i=StrToFloat(StaticText1->Caption);
zero=true;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::Image6Click(TObject *Sender)
{
StaticText1->Caption=FloatToStr(pow(StrToFloat(StaticText1->Caption),3));
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::SpeedButton2Click(TObject *Sender)
{
StaticText1->Caption=FloatToStr(log10(StrToFloat(StaticText1->Caption)));
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::SpeedButton3Click(TObject *Sender)
{
StaticText1->Caption=FloatToStr(log10(StrToFloat(StaticText1-
>Caption))/log10(exp(1)));
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::SpeedButton4Click(TObject *Sender)
{
if (StaticText1->Caption=="0"||zero)
{StaticText1->Caption="";
}
```

```
zero=true;
if(gradus->Checked==True)
{
 StaticText1->Caption=FloatToStr(sin((M_PI/180)*StrToFloat(StaticText1->Caption)));
}
else if(radian->Checked==True)
{
 StaticText1->Caption=FloatToStr(sin((180/M_PI)*StrToFloat(StaticText1->Caption)));
}
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::SpeedButton5Click(TObject *Sender)
{
if (StaticText1->Caption=="0"||zero)
{StaticText1->Caption="";
}
zero=true;
if(gradus->Checked==True)
{
```

```
 StaticText1->Caption=FloatToStr(cos((M_PI/180)*StrToFloat(StaticText1->Caption)));
}
else if(radian->Checked==True)
{
 StaticText1->Caption=FloatToStr(cos((180/M_PI)*StrToFloat(StaticText1->Caption)));
}
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::SpeedButton6Click(TObject *Sender)
{
if (StaticText1->Caption=="0"||zero)
{StaticText1->Caption="";
}
zero=true;
if(gradus->Checked==True)
{
 StaticText1->Caption=FloatToStr(tan((M_PI/180)*StrToFloat(StaticText1->Caption)));
}
else if(radian->Checked==True)
```

```
{
 StaticText1->Caption=FloatToStr(tan((180/M_PI)*StrToFloat(StaticText1->Caption)));
}
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::SpeedButton20Click(TObject *Sender)
{
if (StaticText1->Caption=="0"||zero)
{StaticText1->Caption="0";}
else
{
zero=true;
yigin=true;
yig=yig+StrToFloat(StaticText1->Caption);
}
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::SpeedButton18Click(TObject *Sender)
{
if(yigin==true)
{
```

```
 StaticText1->Caption=FloatToStr(yig);
}
else if(yigin==false)
{
  StaticText1->Caption=FloatToStr(yig*-1);
}
}
//---------------------------------------------------------------------------
void __fastcall TKalkulyator::SpeedButton17Click(TObject *Sender)
{
yig=0;
StaticText1->Caption="0";
}//---------------------------------------------------------------------------
void __fastcall TKalkulyator::SpeedButton14Click(TObject *Sender)
{if (StaticText1->Caption=="0"||zero)
{StaticText1->Caption="0";}
else
{zero=true;
yigin=false;
yig=yig+StrToFloat(StaticText1->Caption);
}}//---------------------------------------------------------------------------
```