`O‘ZBEKISTON RESPUBLIKASI
OLIY VA O‘RTA MAXSUS TA’LIM VAZIRLIGI

Aripov M.M., Otaxanov N.A.

DASTURLASH ASOSLARI

Oliy o‘quv yurtlarining matematika, amaliy matematika va informatika bakalavr yo‘nalishi talabalari uchun o‘quv qo‘llanma

“TAFAKKUR BO‘STONI”

TOSHKENT-2015

Mualliflar: **Aripov Mirsaid** − Mirzo Ulug‘bek nomidagi O‘zbekiston Milliy universiteti “Informatika va tatbiqiy dasturlash” kafedrasi professori, fizika-matematika fanlari doktori;

Otaxanov Nurillo Abdumalikovich – Namangan davlat universiteti “Amaliy matematika va AT” kafedrasi dotsenti v.b., pedagogika fanlari nomzodi.

Taqrizchilar: **Xaydarov Abdug‘affor** – Mirzo Ulug‘bek nomidagi O‘zbekiston Milliy universiteti “Informatika va tatbiqiy dasturlash” kafedrasi professori, fizika-matematika fanlari doktori;

Sagatov Miraziz – Toshkent davlat texnika universiteti “Elektronika va avtomatika” fakulteti “Informatika” kafedrasi professori, texnika fanlari doktori.

Mas’ul muharrir: **Yuldashev Ziyovuddin,** fizika-matematika fanlari doktori, professor.

Mazkur o‘quv qo‘llanmada C++ dasturlash tili haqida boshlang‘ich ma’lumotlar bayon qilingan. Asosiy e’tibor ushbu tilda dastur yozishni boshlash uchun zarur bo‘lgan ma’lumotlarga qaratilgan. Nazariy ma’lumotlarni chuqur o‘zlashtirish uchun qo‘llanmaga yetarli darajada namuna va amaliy ko‘rsatmalar kiritilgan. Har bir mavzu bo‘yicha olingan bilimlarni tekshirish uchun savol va topshiriqlar tizimi keltirilgan.

KIRISH

Insoniyat tarixining koʻp asrlik tajribasi ezgu gʻoyalar, sogʻlom mafkura va zamonaviy bilimlardan mahrum har qanday jamiyat tanazzulga yuz tutishini koʻrsatgan. Shuning uchun bizning mamlakatimiz ham oʻz oldiga ozod va obod Vatan, demokratik jamiyat barpo etish, erkin va farovon hayot qurish, rivojlangan davlatlar qatorida borish maqsadini qoʻygan.

Oʻsib kelayotgan yoshlarni kelajak jamiyatning faol quruvchilari boʻlishi uchun ularni fan va texnikaning eng ilgʻor va zamonaviy yutuqlari bilan qurollantirish hamda olgan bilimlarini amaliyotda qoʻllashga oʻrgatish talab qilinadi. Bu holat ayniqsa, kompyuter bilan bogʻliq iqtisodiy, siyosiy va amaliy masalalarni yechishda yaqqol oʻz aksini topadi.

Demak, yoshlarni zamonaviy kompyuterlar bilan ishlash, xalq xoʻjaligining turli masalalarini yechishga moʻljallangan dasturiy taʼminot bilan muloqot qilishga oʻrgatishdan tashqari, yangi masalalar uchun dasturiy taʼminot ishlab chiqish yoʻl-yoʻriqlari bilan ham tanishtirish lozim boʻladi. Ana shunday dasturiy taʼminot ishlab chiqishning zamonaviy vositalaridan biri C++ dasturlash tili hisoblanadi.

Mazkur tilning ahamiyatini koʻrsatishda bugungi kunda amaliyotga joriy qilinayotgan yangi dasturiy taʼminotlarning kattagina qismi C++ tilida ishlab chiqilayotganligini aytish yetarli deb hisoblaymiz. Shuning uchun ushbu dasturlash tiliga bugungi kunda barcha rivojlangan va rivojlanayotgan davlatlarning oliy taʼlim tizimida katta eʼtibor qaratiladi.

Ushbu qoʻllanma oʻzbek kitobxonlarining talab va takliflariga binoan ishlab chiqilgan boʻlib, unda C++ dasturlash asoslarini oʻrganish uchun zarur boʻlgan boshlangʻich maʼlumotlar va eng asosiy tushunchalar qamrab olingan. Qoʻllanmadagi koʻplab maʼlumotlar birinchi marta oʻzbek tilida bayon qilinmoqda.

“Informatika” o‘quv qo‘llanmasi mualliflarning keyingi yillarda oliy o‘quv yurtlari talabalari orasida “Dasturlash asoslari” hamda “Informatika” fanlari bo‘yicha olib borgan nazariy va amaliy mashg‘ulotlar jarayonida foydalangan ma’lumotlari asosida yuzaga keldi.

C++ dasturlash tiliga bag‘ishlangan o‘zbek tilidagi o‘quv qo‘llanmalari unchalik ham ko‘p emas. Ular asosan bu dasturlash tilining buyruqlar tizimini ochib berishga mo‘ljallangan. Ushbu qo‘llanma esa algoritmlashga yo‘naltirilgan bo‘lib, unda C++ dasturlash tilini o‘rganishning uslubiy jihatlari ochib berilgan hamda qaralayotgan har bir mavzu yuzasidan o‘quvchilarning boshlang‘ich ko‘nikmalarini egallashlari uchun zarur bo‘lgan namunaviy masalalarga katta o‘rin ajratilgan.

Hurmatli o‘quvchi! Ushbu qo‘llanma haqidagi barcha fikr va mulohazalaringizni mirsaidaripov@mail.ru yoki otahanov_n_a@mail.ru elektron manzillari orqali mualliflarga jo‘natishingiz mumkin.

Mualliflar.

I BOB. INFORMATIKANING NAZARIY ASOSLARI

1-§. INFORMATIKA HAQIDA UMUMIY MA’LUMOTLAR

1.1. Informatsiya tushunchasi haqida

XXI asrni “axborotlar asri” deb atash qabul qilingan. Bu bejiz emas. Chunki zamonaviy davlat, jamiyat va insonlarning turli shakl va mazmundagi axborotlarga bo‘lgan ehtiyoji keskin ortib bormoqda. Bunga sabab qilib ularning nafaqat ma’naviy, balki moddiy ahamiyat kasb eta boshlaganini ko‘rsatish mumkin. Amaliyotda axborot tushunchasi bilan parallel ravishda informatsiya yoki ma’lumot tushunchasidan ham keng foydalaniladi. Shuning uchun ham biz umumiylikdan chiqmagan holda, keyingi bayonlarimizda bu tushunchalarni yagona ma’noda qo‘llaymiz.

Informatsiya so‘zi lotincha *informatio* – axborot, izoh, bayon etish ma’nolarida qo‘llaniladi. Bu atama amaliyotda birinchi bo‘lib XX asrning o‘rtalarida K.Shennon tomonidan tor texnik ma’noda, ya’ni aloqa nazariyasi yoki kodlarni uzatish sohasi uchun qo‘llangan.

Informatsiya tushunchasi borasida olimlar o‘rtasida yagona qarash mavjud emas. Bir qator olimlar uni ta’riflashga urinsalar, boshqalari boshlang‘ich tushuncha sifatida qabul qilish lozim, deb hisoblaydilar.

Masalan, S.V.Simonovich informatsiya tushunchasini quyidagicha ta’riflaydi: “Informatsiya – bu axborotlar va ularga mos metodlarning o‘zaro ta’siridir”.

V.A.Kaymin esa bu tushuncha haqida “Informatsiya mazmun jihatidan biror odam, buyum yoki hodisa haqidagi axborot bo‘lsa, shakl jihatidan belgi va signallar yig‘indisidir” degan g‘oyani ilgari suradi.

Axborotlar nazariyasi fani nuqtayi nazaridan informatsiya – bu aynan mavjud noaniqliklarni to‘la bartaraf etuvchi yoki qisman kamaytiruvchi signallardan tashkil topadi. Klod Shennon bu tushunchani “bartaraf etilgan noaniqliklar” deb ta’riflaydi.

Umuman aytganda, informatsiya tushunchasining 30 dan ortiq ta’rifi mavjud bo‘lib, bu ta’riflarning birortasi ham uning tabiatini to‘la ochib bera olmaydi. Shuning uchun ham, boshqa bir guruh olimlar (B.V.Sobol, A.N.Stepanov va h.k.) informatsiya tushunchasini boshlang‘ich tushuncha sifatida qabul qilishni taklif etadilar.

Kibernetika fani asoschisi Norbert Vinerning **“**informatsiya – bu faol harakat, rejalashtirish, boshqarish, ya’ni tizimlarni saqlab qolish, mukammallashtirish va rivojlantirish uchun zarur bilimlar bo‘lib, ***u materiya ham emas, energiya ham emas, u – informatsiya”*** degan fikrlari ham ana shunga ishora qiladi.

Ma’lumki, barcha fanlar (shu jumladan, informatika ham) o‘z nazariyalarini qurishda boshlang‘ich tushunchalardan foydalanadi. Bu tushunchalarning ayrimlarini hech bir ta’rifsiz qabul qilinadi va misollar orqali izohlanadi. Qolgan tushunchalar esa boshlang‘ich va oldindan ta’riflangan tushunchalar yordamida bayon etiladi. Shu nuqtayi nazardan biz ham ma’lumot (informatsiya) tuchunchasini boshlang‘ich sifatida qabul qilish lozim, degan g‘oyani ilgari suramiz. Bu fikr mavjud ko‘plab munozaralarga chek qo‘ysa, ajab emas.

Ikki obyektning o‘zaro ta’siri, qorda qolgan qushning izi, billiard toshlarining bir-biriga urilish tezligi va burchagi, ekinlarning suv va mineral o‘g‘itlarga bo‘lgan ehtiyojlari, korxona omboridagi xomashyo zaxiralari, xodimning ishlagan ish kunlari, talabalarning fanlardan olgan baholari, mashg‘ulotda qatnashayotgan o‘quvchilar ro‘yxati va boshqalarni axborot sifatida qabul qilish mumkin.

Eng sodda ma’lumot alohida signallardan tashkil topadi. Bunday signallar tabiat va jamiyatda cheksiz katta hajmda, shakl va mazmunda, tabiiy yoki sun’iy ravishda mavjud bo‘ladi. Ular insonning sezgi organlari (ko‘rish, eshitish, ta’m bilish, his qilish, hid bilish) yoki maxsus qurilmalar (antennalar, datchiklar va h.k.) tomonidan qabul

qilinishi mumkin. Signallar aynan qandaydir inson, qurilma yoki predmet tomonidan qabul qilinganidan keyingina ma’lumotga aylanadi. Qabul qilingan signal – ma’lumotlar qayta ishlanib, yangi ma’lumotlarga aylantirilishi mumkin. Shunga ko‘ra ma’lumotlar birlamchi, ikkilamchi va h.k. tarzida bo‘ladi.

Masalan, tuproqda qoldirilgan signal – izni ovchi ko‘rganidan keyingina tirikchilik yoki hayot-mamot uchun zarur bo‘lgan ma’lumotga aylanadi, aks holda u shunchaki iz bo‘lib qolaveradi.

Ko‘rinib turibdiki, har qanday signal ma’lumot shaklini qabul qilishi uchun inson, qurilma yoki predmetlar bilan o‘zaro ta’sirga kirishishi lozim. Boshqacha aytganda, signallar manbasi va foydalanuvchisi (qabul qiluvchisi) bo‘lgandagina ma’lumotga aylanadi.

1.2. Ma’lumotlarning xossalari

Yuqorida ta’kidlanganidek, ma’lumot (informatsiya, axborot) tushunchasi turli fanlar doirasida turlicha talqin qilinadi va bir qator xossalarga ega bo‘ladi. Biz qarayotgan holda, ma’lumotlar uchun eng muhim xossalar quyidagilardan iborat bo‘ladi: dualizm (ikkiyoqlamalik), to‘liqlik, ishonchlilik, xoslik (adekvatlik), mavjudlik, dolzarblik.

Ma’lumotlarning dualligi ularning ikkiyoqlama ekanligini tavsiflaydi. Bir tomondan, berilgan obyektiv ma’lumotlarga ko‘ra u obyektiv, ikkinchi tomondan qayta ishlashda qo‘llanadigan usullarning subyektivligiga ko‘ra u subyektiv. Ma’lumotlarni qayta ishlashga mo‘ljallangan metodlar ma’lumotlarni u yoki bu darajada o‘zgartirib, ma’lum bir subyektiv omillarni yuzaga keltirishi mumkin. Masalan, ikki kishi aynan bir xil kitobni o‘qishi va butunlay boshqa-boshqa xulosalar chiqarishi mumkin.

Ma’lumotlarning to‘liqligi biror qarorni qabul qilish yoki yangi ma’lumotlarni shakllantirish uchun yetarlilik darajasini belgilaydi. Ma’lumotlarning to‘liq bo‘lmasligi ko‘plab noaniqliklarning yuzaga kelishiga sabab bo‘lishi mumkin. Ma’lumotlarning o‘ta ko‘p bo‘lishi ular orasidan eng muhimlarini tanlab olish jarayonida muammolar paydo qiladi. Bajarilayotgan ishga qo‘shimcha ravishda tartiblash,

tabaqalash, navlarga ajratish kabi masalalarni hal qilishga olib keladi. Koʻrinib turibdiki, maʼlumotlarning toʻliq boʻlmasligi yoki ortiqcha boʻlishi toʻgʻri va oʻziga xos qarorlarni qabul qilishni qiyinlashtirib yuboradi.

Maʼlumotning ishonchliligi deganda uning haqiqatga yetarlicha darajada mos kelishi nazarda tutiladi. Maʼlumotlar toʻliq boʻlmaganda ularning ishonchligi ehtimollik yordamida tavsiflanadi. Masalan, tanga tashlanganda gerbli tomonning tushish ehtimoli 50 foizni tashkil qiladi.

Maʼlumotlarning xosligi (adekvatligi) qaralayotgan real jarayon yoki obyektga mos kelishi bilan xarakterlanadi. Albatta, ideal maʼnodagi xoslikka erishib boʻlmaydi, chunki koʻpincha yetarli darajada toʻliq boʻlmagan maʼlumotlar bilan ishlashga toʻgʻri keladi. Ularni qayta ishlashda qoʻllanadigan metodlarning yetarli boʻlmasligi xos maʼlumotlar olishni mushkullashtiradi.

Maʼlumotning mavjudligi ehtiyoj tugʻilganda zarur boʻlgan barcha maʼlumotlarni olish imkoniyatining mavjud boʻlishini anglatadi. Mavjudlik maʼlumotlarga ham, ularni qayta ishlash uchun moʻljallangan metodlarga nisbatan ham qaraladi. Bu komponentalarning birortasining mavjud boʻlmasligi xos boʻlmagan maʼlumotlarning yuzaga kelishiga sabab boʻladi.

Joriy vaqtda ***dolzarb*** boʻlgan maʼlumot biroz fursat oʻtganidan keyin dolzarbligini yoʻqotishi mumkin. Masalan, bugungi telekoʻrsatuvlar dasturining ertaga hech kimga kerak boʻlmasligi hammaga yaxshi maʼlum.

Maʼlumotlarning tushunarliligi taklif etilayotgan maʼlumotlar foydalanuvchilarga qay darajada tushunarli ekanligini belgilaydi. Masalan, bir tilda tushunarli boʻlgan sodda koʻrsatmalar boshqa tilda muomala qiluvchi kishilar uchun tushunarli boʻlmasligi mumkin.

Maʼlumotning ahamiyati uning qaralayotgan masala doirasida tutgan oʻrni bilan belgilanadi. Bitta foydalanuvchi uchun oʻta muhim boʻlgan maʼlumotlarning ikkinchisi uchun hech qanday ahamiyati boʻlmasligi mumkin. Masalan, Namangan shahrining bugungi ob-

havosi haqidagi ma’lumotlar Nyu-York shahri aholisi uchun hech qanday qiymatga ega bo‘lmaydi.

Ma’lumotlar quyidagi alomatlariga ko‘ra ahamiyatga ega bo‘lishi mumkin: semantik, sintaktik, pragmatik.

Semantik (ma’noviy) – ma’lumotlarni ular ifodalayotgan obyekt qiyofasiga qay darajada mos kelishini aniqlaydi. Semantik ahamiyat informatsiyaning ma’noli mazmunini hisoblash bilan belgilanadi. Bunda informatsiya aks ettirgan ma’lumotlar tahlil qilinadi, ma’nolar bog‘liqligi ko‘riladi. Bu shakl informatsiya xususida tushuncha va tasavvurlarni shakllantirish, ma’nosi, mazmunini aniqlash, umumlashtirishga xizmat qiladi.

Sintaktik (hajmiy) **–** ma’lumotning mazmuniga tegmagan holda, uning rasmiy strukturaviy xarakteristikalarini ifodalaydi. Sintaktik darajada ifodalash usulida informatsiyani eltuvchi turi, uzatish va qayta ishlash tezligi, ifodalash kodining o‘lchamlari, bu kodlarni o‘zgartirish aniqliligi va ishonchliligi hisobga olinadi.

Pragmatik (foydalanuvchanligi) – informatsiya bilan foydalanuvchi munosabatlarini aks ettiradi, boshqarilayotgan tizim maqsadiga taklif etilayotgan ma’lumotlarning muvofiqligini ifodalaydi. Informatsiyaning pragmatik xususiyatlari faqat informatsiya (obyekt), foydalanuvchi va boshqarish maqsadlarining umumiyligida namoyon bo‘ladi.

1.3. Informatika fani va uning vazifalari haqida

Ixtiyoriy davlat, jamiyat yoki inson faoliyatini yig‘iladigan, qayta ishlanadigan, uzatiladigan ma’lumotlarsiz tasavvur qilishning umuman iloji yo‘q. Davlat boshqaruv jarayonida, jamiyat faoliyatini tashkil qilishda, insonlar hayotida turli shakl va mazmundagi bunday ma’lumotlarning ahamiyatini baholab bo‘lmaydi.

Akademik B.V.Sobolning ta’kidlashicha, XVII asrdan boshlab, har 20 yilda ilmiy ma’lumotlarning hajmi ikki baravar oshgan. Bugungi kunda esa bunday ko‘rsatkichga har 5-6 yilda erishilmoqda. Zamonaviy mutaxassislar o‘z faoliyatlarini me’yorda tashkil etish uchun 80

foizigacha ish vaqtlarini sohalariga xos yangiliklarni o‘rganishga sarflashmoqda[1].

Demak, bunday katta hajmdagi ma’lumotlar bilan ishlash uchun maxsus qonun-qoidalarni ishlab chiqish lozim bo‘ladi. Bu masala informatika fanining eng asosiy masalasi hisoblanadi.

Informatika (*inform* – informatsiya, *matics* – haqidagi fan) tushunchasi asli XVII asrdagi kutubxonashunoslikka oid atama bo‘lib, bugungi kundagi ma’noda XX asrning 80-yillaridan muomalaga kirgan. G‘arb davlatlarida *Computer Science* – kompyuterlar haqidagi fan nomi bilan yuritiladi.

Olimlar o‘rtasida informatika faniga bo‘lgan qarashlar turlicha bo‘lib, ularning har biri fanning faqat ma’lum bir jihatlarini qamrab oladi, xolos. Masalan, akademik A.P.Ershov “Informatika fani ma’lumotlarni yig‘ish, saqlash, qayta ishlash va uzatish qonun-qoidalarini o‘rganadi”, deb yozgan edi.

G.A.Ojegov informatika faniga shunday ta’rif beradi: “Informatika – bu ma’lumotlarning umumiy xususiyatlari va strukturasi, shuningdek, ularni yig‘ish, saqlash, qayta ishlash, uzatish hamda faoliyatning turli ko‘rinishlarida foydalanish bilan bog‘liq masalalarni o‘rganuvchi fandir”.

S.V.Simonovich informatika fani haqida shunday deydi: “Informatika – texnik fan bo‘lib, ma’lumotlarni hisoblash texnikasi vositasida yaratish, saqlash, qayta ishlash usullarini tizimlashtiruvchi hamda bu vositalarni boshqarish va ishlash prinsiplarini o‘rganadi”.

B.V.Sobol esa bu fanni boshqacharoq ta’riflaydi. Informatika quyidagi masalalarni o‘rganuvchi fandir:

- hisoblash texnikasi vositalari (HTV) yordamida hisoblash jarayonlarini amalga oshirish usullari;
- HTVning strukturasi, tarkibi va ishlash prinsiplari;
- HTV ni boshqarish prinsiplari.

Bizningcha, zamonaviy informatika fanining maqsad va vazifalaridan kelib chiqib, uni quyidagicha ta’riflagan to‘g‘riroq bo‘lar edi:

[1] Соболь. Б.В. Информатика. –Ростов на Дону, 2007. -С. 9.

informatika − ma’lumotlarni kompyuter yordamida yig‘ish, saqlash, qayta ishlash, foydalanish va uzatish qoidalarini o‘rganuvchi fan hisoblanadi.

Informatika fani ma’lumotga o‘zaro bog‘langan, atrof-muhit obyektlari yoki holatlari, sodir bo‘layotgan voqea va jarayonlar haqidagi signallar, xabarlar va tushunchalar sifatida qaraydi.

Ma’lumotlarning informatsiya manbasidan foydalanuvchiga uzatilishini ta’minlovchi yo‘l va jarayonlar ***informatsion kommunikatsiyalar*** deb yuritiladi

Ta’rifdan ko‘rinib turibdiki, informatika – ko‘plab fanlar doirasida erishilgan yutuqlardan foydalanuvchi nazariy va amaliy fan bo‘lib, u nafaqat o‘rganuvchi, balki taklif ham etuvchi fan ham hisoblanadi. Bu ma’noda informatikani texnologik fan ham deb qarash mumkin. U o‘z vazifalarini kibernetika, avtomatika, informatsiyalar nazariyasi, statistika, kodlash nazariyasi, matematik mantiq, algoritmlar nazariyasi, hujjatlashtirish kabi fanlar bilan uzviy aloqada amalga oshiradi.

Informatika fani tarkibini shartli ravishda uchta katta bo‘limga ajratish mumkin:

Informatsion jarayonlar:

Nazariy asoslari: kodlash nazariyasi, informatsiyalar nazariyasi, graflar nazariyasi, to‘plamlar nazariyasi, matematik mantiq va h.k.

Amaliy asoslari: ma’lumotlarning tuzilmalari, o‘lchamlari, ma’lumotlarni kodlash, ixchamlash va h.k.

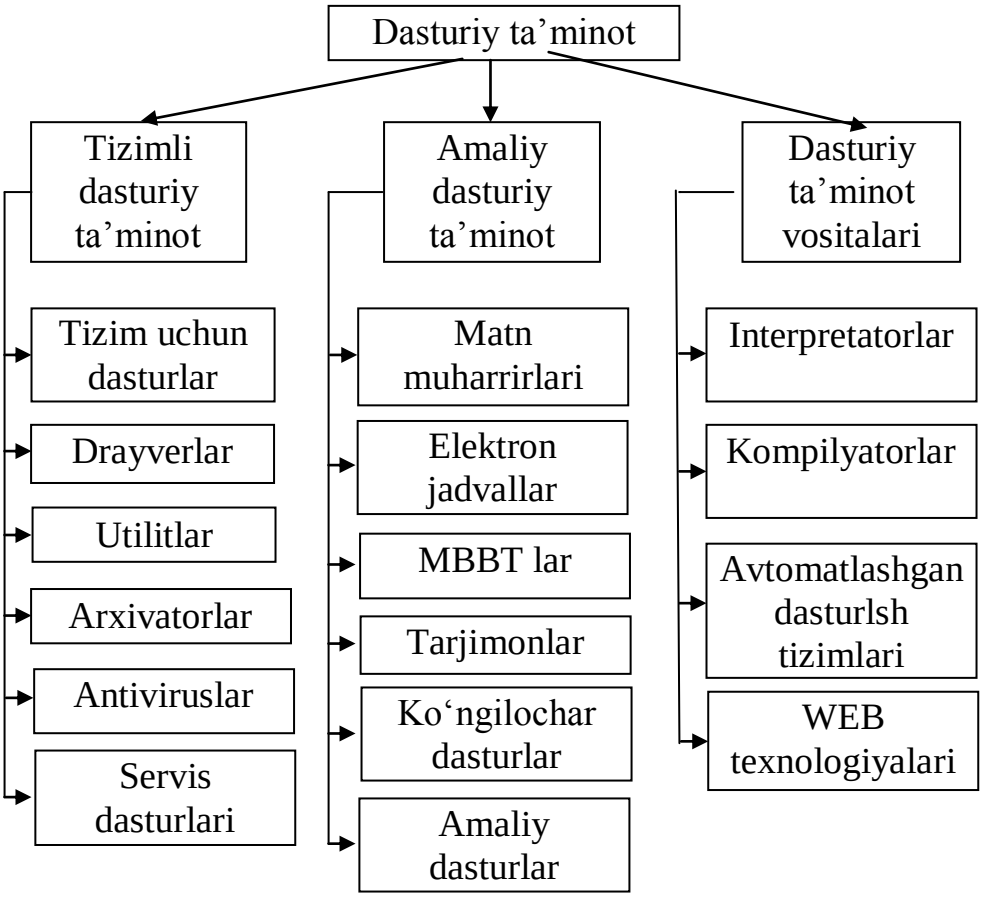
Texnik ta’minot:

Nazariy asoslari: matematik mantiq, elektronika, avtomatika, kibernetika va h.k.

Amaliy asoslari: raqamli qurilmalar sintezi, hisoblash texnikalari arxitekturasi, hisoblash texnikasining qurilmalari, kompyuter tarmoqlari uchun qurilmalar va h.k.

Dasturiy ta’minot:

Nazariy asoslari: algoritmlar nazariyasi, matematik mantiq, graflar nazariyasi, o‘yinlar nazariyasi, kompyuter lingvistikasi va h.k.



Amaliy asoslari: tizimli dasturlar, amaliy dasturiy mahsulotlar, interfeyslar, yordamchi dasturlar va h.k.

Dasturiy ta’minotning tashkil etuvchilarni 3 guruhga bo‘lish mumkin.

Takrorlash uchun savol va topshiriqlar

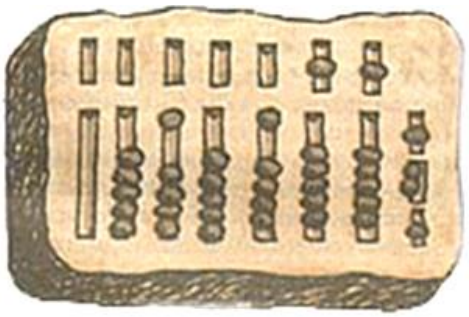
1. Informatsiya tushunchasi nima?
2. Informatsiyaning asosiy xossalarini ayting.
3. Ma’lumotning ahamiyati qanday alomatlarga ko‘ra aniqla-nadi?
4. Informatika fani nimani o‘rganadi?
5. Informatika fanining tarkibi nimalardan iborat?
6. Kompyuterlarning dasturiy ta’minotini tashkil etuvchilarni ayting.

2-§. HISOBLASH TEXNIKASINING RIVOJLANISH TARIXI

2.1. Ilk hisoblash qurilmalari

Qadimdan odamlarda zaxira to‘plash, o‘ljani taqsimlash, mol ayriboshlash kabi masalalarini hal qilishda sanash yoki hisoblashga ehtiyoj bo‘lgan. Bu jarayonda ular barmoqlar, tayoqchalar, toshlar va hatto, arqonlardan foydalanishgan. Arxeologik qazishmalar natijasida topilgan tugunli arqonchalar, kemtilgan tayoqcha va suyaklar o‘sha davr odamlari hisoblash ishlarni bajarishda ana shunday vositalardan foydalanganliklariga ishora qiladi.

Hisob arqoni va taxtachasi



“Abak” taxtasi

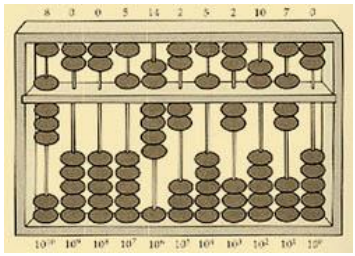
Yechish talab qilinayotgan masalalarning murakkablashuvi hisoblashning yangi usullari va vositalariga ehtiyojning paydo bo‘lishiga olib keldi. Tarixan har bir davlatning o‘z sanoq sistemasi, uzunlik, og‘irlik va hajmlarni o‘lchash usullari, pul birliklari shakllangan. Savdo-sotiq, ilm-fanning rivojlanishi bu ma’lumotlarni bir davlatdagi shakldan ikkinchi davlatdagi shaklga o‘tkazishni taqozo etishi yangi hisoblash qurilmasini ishlab chiqishga turtki bergan. Ulardan biri eramizdan avvalgi VI-V asrlarda yaratilgan va “Abak” nomi bilan mashhur bo‘lgan.

Qadimdan har bir sterjenida 5 tadan tosh o‘rnatilgan cho‘tlardan foydalanishgan. XVII asrdan boshlab Qadimgi Rus davlatida bu cho‘tning mukammallashtirilgan varianti hisoblangan Rus cho‘tlaridan foydalanish udum bo‘lgan.

850-yillarda atoqli o‘zbek olimi, matematik Muhammad ibn Muso al-Xorazmiyning o‘nli sanoq sistemasi uchun nol raqamini kiritishi hamda shu sistemada barcha arifmetik amallar bajarishning umumiy qoidalarini “Al-jabr va al-muqobala” kitobida yozib qoldirishi hisoblash mashinalarining yaratilishida eng kuchli zamin bo‘lib xizmat qildi. Chunki bu olimga qadar biror kishi o‘nlik sanoq sistemasida amallarni bajarish tartibini to‘liq bayon etmagan edi. Al-Xorazmiyning yana boshqa bir kitobida hind arifmetikasi haqida batafsil ma’lumotlar keltirilgan. 300 yildan so‘ng ushbu kitoblar lotin tiliga tarjima qilindi. Shu tariqa Yevropa davlatlarida Xorazmiy ishlab chiqqan qoidalar keng joriy etila boshlandi. “Algoritm” so‘zi al-Xorazmiy nomi bilan bevosita bog‘liq.

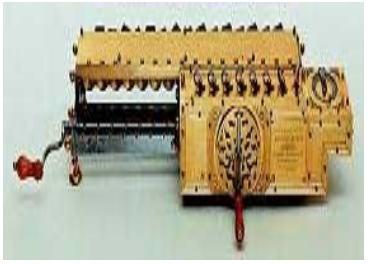
Matematika fanining keskin rivojlanishi XVII asrda yangi hisoblash qurilmasiga ehtiyojni paydo qildi va bunga javoban 1642-yilda Blez Paskal tomonidan 8 xonagacha bo‘lgan sonlar ustida qo‘shish va ayirish amallarini bajara oladigan hisoblash mashinasi ishlab chiqildi. Uning o‘lchamlari 36×13×8 sm bo‘lib, foydalanish uchun juda ham qulay hisoblangan.

1673-yilda nemis matematigi Vilgelm Leybnits tomonidan to‘rt amalni bajarishga moslashgan va o‘lchamlari 100×30×20 sm bo‘lgan mexanik qurilma ishlab chiqildi. Bu qurilma hisoblash ishlarini shesternali g‘ildiraklar yordamida amalga oshirgan.



5 toshli Xitoy cho‘ti

Paskal mashinasi

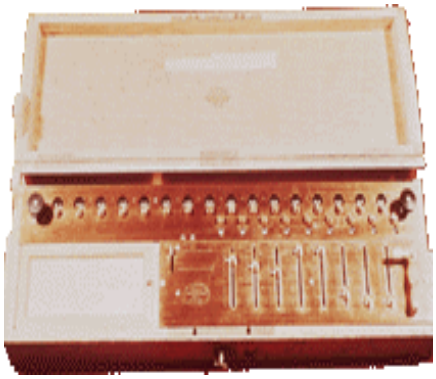


Leybnits mashinasi

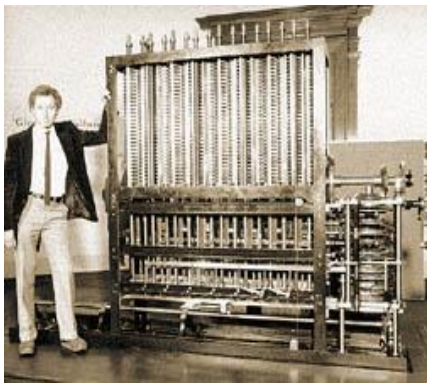
1822-yilda Ch.Bebbidj tomonidan ishlab chiqilgan hisoblash mashinasi kvadratlar jadvali, $y = x^2 + x + 41$ funksiyasining qiymatlarini hamda bir qator jadvallarni hisoblash imkoniga ega edi. Ammo

bu mashina oxirigacha yetkazilmadi va hozirda London Qirollik kolleji muzeyida saqlanadi. Ch.Bebbidj bundan umidsizlikka tushmadi va inson yordamisiz boshqariladigan analitik mashina ustidagi ishlarini boshlab yubordi. Ammo o‘sha vaqtda texnikaning yetarli taraqqiy etmaganligi bu g‘oyaning ham oxiriga yetishiga yo‘l qo‘ymadi. Olimning asosiy xizmati shundaki, u birinchi bo‘lib hisoblashlarni dastur yordamida bajarish g‘oyasini taklif etdi va qisman bu g‘oyani amalga oshirishga erishdi.

Dunyoda birinchi mexanik kalkulyator 1818-yilda elzaslik Charlz Ksaver Tomas (1785-1870) tomonidan ishlab chiqildi. Bu kalkulyator to‘rt arifmetik amalni bajarish imkoniga ega bo‘lgan va 1822-yildan sanoat korxonasida ishlab chiqish yo‘lga qo‘yilgan. Shunday qilib, Ch.Tomas hisoblash mashinalarini ishlab chiqish sanoatiga asos soldi.



Birinchi kalkulyator



Bebbidjning analitik mashinasi

XX asr boshlarida konstruktorlar hisoblash mashinalarida elektromagnit relelardan foydanish mumkinligiga e’tibor qaratishdi. 1941-yilda nemis injeneri ana shunday relelar yordamida ishlaydigan mashinani yaratishga muvaffaq bo‘ldi. Shu vaqtning o‘zida 1943-yilda amerikalik Govard Eyken IBM firmasida Ch.Bebbidj g‘oyalari asosida ishlaydigan “Mark-1” hisoblash mashinasini qurishga erishdi. Bu mashina ulkan (uzunligi 15 m, balandligi 2,5 m) bo‘lib, 800 mingta qismdan iborat va konstantalar uchun 60 ta, qo‘shish uchun esa 72 ta xotira registrlarini, ko‘paytirish va bo‘lish uchun markaziy

bloklarni o‘z ichiga olgan. Mazkur mashina 23 xonagacha bo‘lgan o‘nlik sanoq sistemasidagi sonlar ustida amallarni bajarish imkoniga ega bo‘lib, qo‘shish amalini bajarish uchun 0,3 sekund, ko‘paytirish uchun esa 3 sekund vaqt sarf qilgan.

2.2. Elektron hisoblash mashinalari

Dastlabki EHMlar ustida tadqiqotlar 1937-yilda, AQSHda professor Jon Atanasov rahbarligida boshlangan bo‘lib, bu mashinalar asosan matematik-fizik masalalarga yoki sodda qilib aytganda, faqat hisob-kitob bilan bog‘liq masalalarga mo‘ljallangan edi. Ammo Atanasovning bu loyihasi to‘laligicha oxiriga yetmadi. Shunday bo‘lsada, bu ish o‘ttiz yildan so‘ng bir necha sud majlislarida ko‘rib chiqildi va olim EHMlar asoschisi deb tan olindi.

1942-yilda pensilvaniyalik fizik Jon Mouchli (1907-1980) Atanasovning loyihasi bilan tanishib chiqib, “Tez ishlaydigan elektron qurilmalardan hisoblash ishlarida foydalanish” mavzusidagi loyihasini taqdim qilgan. 1943-yilda Mouchli va Jon Ekkert rahbarligi ostida ENIAC (Electronic Numerical Integrator and Computer − Elektronli hisoblagich integratori va kalkulyator) mashinasi ustida tadqiqotlar boshlandi. 1945-yilda loyihaga mashhur matematik Jon fon Neyman taklif qilindi. Uning asosiy xizmati shundaki, EHMlar uchun umumiy bo‘lgan tuzilmani tavsiya etdi. Quyida ulardan ayrimlarini keltiramiz:

- mashina boshqaruv, arifmetik, xotira va aloqa (operator bilan) qurilmalaridan iborat bo‘lishi lozim;
- dastur ham, boshlang‘ich ma’ulmotlar kabi EHM xotirasida joylashadi;
- mashina sonli ma’lumotlarni sonli ko‘rinishda yozilgan buyruqlardan farqlay olishi kerak;
- tarkibida kiritish va chiqarish qurilmalarining bo‘lishi shart.

ENIAC mashinasini ishlab chiqishda bu g‘oyalar to‘la amalga oshirildi. Loyihada ko‘zda tutilgan ishlar 1946-yilda yakunlanib, shu yilning 16-fevralida tarixda birinchi elektron hisoblash mashinasi

bo‘lgan ENIAC namoyish qilindi. So‘ngra bu mashina Aberdin poligoniga olib o‘tildi va u yerda undan 1955-yilgacha foydalanildi.

ENIAC mashinasi haybatli o‘lchamga ega edi. U o‘z ichiga 18 mingta elektron lampani olgan, 90×15 m maydonni egallagan, og‘irligi 30 t bo‘lib, 150 kVt elektr energiyasini iste’mol qilgan va 100 kGs chastota bilan ishlagan. Qo‘shish amali uchun 0,2 ms, ko‘paytirish uchun esa 0,28 ms vaqt sarflagan.

Shu vaqtdan boshlab hozirgacha ishlab chiqilgan EHMlar shartli ravishda to‘rtta avlodga ajratiladi. Bu avlodlar bir-biridan EHMning elementlar bazasi hamda imkoniyatlari bilan farq qiladi.

1-avlod hisoblash mashinalari. Bu mashinalar elektron lampalar yordamida ishlagan. Dastlabki EHM lar seriyali tarzda 1951-yildan boshlab ishlab chiqarila boshlangan va UNIVAC (Universal automatic computer) deb nomlangan. Bu mashinalarning takt chastotasi 2,25 MGs bo‘lib, 5000 lampani o‘z ichiga olgan.

Sobiq ittifoqda birinchi EHM 1949-yilda akademik S.A.Lebedev rahbarligi ostida Elektrotexnika istitutining Kiev shahridagi Hisoblash texnikasi va modellashtirish laboratoriyasida MESM nomi bilan ishlab chiqilgan.

Birinchi yapon EHMi “Fudjik” 1956-yilda ishlab chiqilgan. Germaniya bunday ishlarni 1958-yildan boshlab yo‘lga qo‘ygan.

2-avlod hisoblash mashinalari. XX asrning 50-yillarida ayrim firmalar tranzistorli EHMlar bo‘yicha ishlar boshlaganlarini e’lon qilishdi. 1955-yilda AQSHda 800 ta tranzistor va 11 ming germaniy diodi yordamida ishlaydigan TRADIC raqamli kompyuteri namoyish qilindi. To‘la tranzistorlardan tashkil topgan “Philco-2000” EHMi haqida 1958-yilning noyabrida e’lon qilindi. U 56 ming tranzistor va 1200 diodni o‘z ichiga olgan bo’lib, tarkibida hamon 450 ta elektron lampa mavjud edi. “Philco-2000” mashinasi qo‘shish amalini 1,7 mks, ko‘paytirishni esa 40,3 mks vaqt mobaynida bajarish imkoniga ega bo‘lgan.

1958-yilda Angliyada tranzistorli EHMlarning birinchisi – ”Elliot-803”, Germaniyada – “Simens-2002”, Yaponiyada H-1 ishlab chiqildi.

Sobiq ittifoqda birinchi yarim o‘tkazgichli EHM ustidagi ishlar Yerevan shahrida E.L.Brusilovskiy rahbarligi ostida 1960-yilda yakunlandi. Bu EHM tarixda “Razdan-2” nomini oldi.

1960-yilda IBM firmasi qudratli “Stretch” (IBM-7030) hisoblash tizimini ishlab chiqdi. Bu tizim tarkibidagi 169 ming tranzistor evaziga avvalgilaridan 100 marta tezroq ishlagan, taktotalar chastotasi esa 100 MGs ni tashkil qilgan.

Control Data firmasi ishlab chiqqan CDC-6600 kompyuterida birinchi bo‘lib ma’lumotlarni ko‘pprotsessorli qayta ishlash mexanizmi yo‘lga qo‘yilgan. Bu kompyuter o‘z ichiga ko‘p sonli arifmetik-mantiqiy qurilmalar va 10 ta periferik protsessorni olib, amallarni bajarish tezligini sekundiga 3 mln taga yetkazgan.

1964-yilda Penza shahrida “Ural” seriyasidagi EHMlarni ishlab chiqarish yo‘lga qo‘yildi. Shu davrda ishlab chiqilgan va yuqori darajada ishonchli sanalgan MESM-6 mashinasi 60 ming lampa va 200 ming dioddan tashkil topgan bo‘lib, sekundiga 1 mln gacha amallarni bajara olgan.

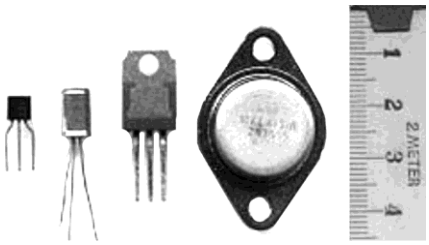
3-avlod hisoblash mashinalari. Amerikalik olimlar D.Kilbi va R.Noyslar bir vaqtning o‘zida integral sxemalarni ishlab chiqishdi. Integral sxema maxsus nisbatda olingan magniy va kremniy qatlamlaridan iborat bo‘lib, bir necha o‘n minglab yarim o‘tkazgich vazifasini bajarish imkoniga ega edi. Fandagi bu yutuq EHM ishlab chiqarish sohasiga tatbiq etildi. Integral sxemalarni ommaviy ravishda ishlab chiqarish 1962-yildan boshlangan.

Integral sxemalar yordamida ishlaydigan dastlabki kompyuter IBM-360 ni 1964-yilda IBM firmasi taqdim etdi.

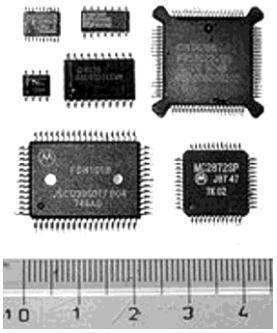
Integral sxemalar asosidagi birinchi sobiq ittifoq kompyuteri “Nairi-3” 1970-yildan boshlab ishlab chiqarilgan. Mashhur ES-1010 EHM esa 1972-yildan ishlab chiqarilgan. Bu mashinalarning amallarni bajarish tezligi sekundiga 2 mln gacha yetkazilgan.

AQShda 1972-yilda o‘z ichiga 64 ta ma’lumotni qayta ishlovchi qurilmalarni olgan ILLIAC-IV mashinasi yaratildi. Uning amallarni bajarish tezligi sekundiga 200 mln gacha yetdi.

ILLIAC-IV mashinasi



Tranzistorlar



Interal sxema

4-avlod hisoblash mashinalari. XX asrning 70-yillari katta integral sxemalar asosida ishlaydigan kompyuterlarga o‘tish bilan ajralib turadi.

Edvard Xoff tomonidan katta EHMlarning markaziy protsessori vazifasini bajara oladigan yangi mikrosxema taklif qilindi. Katta integral sxemalar deb ataluvchi bunday mikrosxemalar katta sondagi integral sxema, tranzistor va boshqa elektron elementlardan tashkil topgan bo‘ilb, to‘rtinchi avlod EHMlarini ishlab chiqarishda asos bo‘lib xizmat qildi.

1971-yilda protsessor imkoniyatlarini ayrim amallar bilan cheklash, mikroprogrammalarni esa doimiy xotiraga oldindan yozib qo‘yish g‘oyasi taklif etildi. Hajmi 16 kb bo‘lgan doimiy xotira qurilmasini qo‘llash 200 tagacha oddiy integral sxemalardan voz kechish uchun yo‘l ochadi. Bu vaqtda oddiy mikroprotsessorning 1 kv. mm yuzasida 500 tagacha tranzistorlar mavjud bo‘lar edi.

1971-yilda INTEL firmasi birinchi mikroprotsessor Intel-4004 mikrosxemasini ishlab chiqdi. Bu mikroprotsessor asosida dastlabki elektron kalkulyatorlar yuzaga keldi. 1972-yilda matnlarni qayta ishlash uchun mo‘ljallangan 8 razryadli Intel-8008 mikroprotsessori ishlab chiqarilishi shaxsiy kompyuterlarning ishlab chiqarilishiga turtki berdi.

70-yillarning o‘rtasiga kelib, kompyuter bozorida ikki yo‘nalish boshqalaridan yaqqol ajralib qoldi. Bu yo‘nalishlarning biri superkompyuterlar, ikkinchisi esa shaxsiy kompyuterlarni ishlab chiqarishda o‘z aksini topdi.

Superkompyuter yo‘nalishi bo‘yicha ishlab chiqarilgan ko‘p protsessorli kompyuterlaridan biri “Elbrus” bo‘lib, uning protsessori sekundiga 200 mlndan ortiq amallarni bajara olgan. Hozirgi paytda juda ko‘p mamlakatlarda mavjud bo‘lgan va tezligi bir necha yuzlab milliard amallarni tashkil qiladigan kompyuterlarni ham super kompyuterlar jumlasiga kiritish mumkin.

Apple-I deb nomlangan birinchi shaxsiy kompyuter Stiven Jobs va Stiv Voznyak tomonidan 1976-yilda qurildi. Bu kompyuterning arxitektura va modullar tizimining ochiq deb e’lon qilinishi uning jadallik bilan rivojlanishi va jahon bozorini egallashiga sabab bo‘ldi.

Kuchli raqobat INTEL firmasi tomonidan 1982-yilda Intel-80286 protsessorining ishlab chiqarilishiga olib keldi. Bu protsessor 2 Gbaytgacha virtual xotira bilan ishlash imkoniyatiga ega bo’lgan.

Hozirgi vaqtda yaratilgan shaxsiy kompyuterlar xotira sig‘imi, amallarni bajarish tezligi va boshqa ko‘rsatkichlari bo‘yicha katta va mini EHMlardan pastroq tursa ham, shunday yutuqlarga egaki, arzon narxi, ishonchliligi, gabarit o‘lchamlarining kichikligi, ishlab chiqarish va ekspluatatsiya jarayonida oddiyligi bilan boshqa turdagi EHMlardan tubdan ajralib turadi.

5-avlod hisoblash mashinalari. Hozirgi vaqtda sanoati rivojlangan ko‘plab davlatlar hisoblash texnikasining 5-avlodini, sifat jihatidan mutlaqo yangi, foydalanuvchilar uchun qulay hisoblash sistemalarini yaratish ustida ishlamoqdalar. Ko‘plab loyihalar mavjud, ammo ularning birortasi ham oxirigacha yetganicha yo‘q. 5-avlod EHMlarida mashina tillarini tabiiy tillarga yaqinlashtirish (matn, nutq, tasvir va boshqalar) ustida ham muntazam ishlar olib borilmoqda. Bundan tashqari, 5-avlod EHMlari yordamida hisoblash sistemasi tashkil qilinganda ularni foydalanuvchilar uchun intellektual abonent punkti sifatida qo‘llashni ham ko‘zda tutish kerak. Bu avlod mashinalarini inson faoliyatida qo‘llash va intellektual informatsiya ishlab chiqarish jarayonini boshqarishda foydalanish optimal qarorlar qabul qilishga jiddiy va samarali ta’sir ko‘rsatishi mumkin.

5-avlod EHMlarining o‘ziga xos xususiyatlari quyidagilardan iborat:

- axborotlarni har tomonlama aniqlangan va formallashgan bilim sifatida qayta ishlash;

- bajarish mumkin bo‘lgan funksiyalar sinfini kengaytirish;

- yuqori darajada ixtisoslashuvi;

- kompyuterlar yordamida yechilishi rejalashtirilgan har bir masala uchun xos bo‘lgan bilimlar bazasini yaratish va ularni amaliyotga joriy qilish;

- intellektual interfeys vositalaridan iloji boricha to‘la foydalanish hamda kompyuter bilan muloqotni osonlashtirish va masalalarni qo‘yish va hal qilishda oddiy insoniy tillardan foydalanishga erishish.

5-avlod EHMlari keng foydalanuvchilar ommasiga mos keladigan va sodda bo‘lishi uchun yuqorida aytganimizdek EHM bilan muloqotni tabiiy tilda, shuningdek, grafiklarni kiritish-chiqarish, hujjat, qo‘lyozma belgi va boshqalarni kiritish yoki o‘qish maxsus qurilmalar orqali amalga oshirilishi kerak. Inson va mashina-ning o‘zaro aloqa jarayonida optimal dialog rejimini rivojlantirish kun tartibidagi asosiy masalalardan biri bo‘lib turibdi.

Takrorlash uchun savol va topshiriqlar

1. Hisoblash texnikasi qachondan boshlab rivojlana boshlagan?
2. Al-Xorazmiyning fanga qo‘shgan ulkan hissasi nimadan iborat?
3. Ilk mexanik hisoblash texnikalari va ularni ishlab chiqqan olimlarn ismlarini ayting.
4. Birinchi EHM qachon va kim tomonidan yaratilgan?
5. EHMlarning qanday avlodlarini bilasiz? Ular bir-biridan qanday tomonlari bilan farq qiladi?

3-§. KOMPYUTERNING ASOSIY QURILMALARI

3.1. Umumiy ma’lumotlar

Kompyuter – murakkab hisoblash tizimi bo‘lib, inson tafakkuriga nisbatan million, hatto milliard marta tezroq hisoblash hamda mantiqiy yechimlarni qabul qilish qurilmasi sanaladi.

Kompyuter o‘ta itoatkor qurilma bo‘lib, inson tomonidan berilgan buyruqlarni tez, sifatli va aniq bajara oladi. Kompyuterlar bunday buyruqlarni bajarishi uchun ularda mos dasturiy ta’minot o‘rnatilgan bo‘lishi zarur.

Kompyuterlardan foydalanuvchilarni ikki guruhga ajratish mumkin: oddiy foydalanuvchilar va amaliy dasturchilar.

Amaliy dasturchilar fan, turli xo‘jalik sohalari va hayotda uchrab turadigan masalalar uchun amaliy dasturlar ishlab chiqish bilan shug‘ullanadilar. Ular tomonidan kompyuterda hal qilish mumkin bo‘lgan juda ham katta sinfdagi masalalar uchun dasturiy ta’minotlar yaratilgan. Oddiy foydalanuvchilar esa amaliy dasturchilar mahsulotlaridan turli maqsadlarda foydalanadilar.

Kompyuter tizimiga kiruvchi turli qurilmalar uning apparat ta’minoti, kompyuter tomonidan bajarilishi nazarda tutilgan dasturiy majmualar esa dasturiy ta’minoti deb ataladi.

3.2. Kompyuter turlari va asosiy qurilmalari haqida

Bugungi kunda amaliyotda turli xil kompyuterlardan keng foydalaniladi. Ular bir-biridan asosan bajaradigan amallarining soni va arxitekturasi bilan farqlanadi. Kompyuterlarni bunday imkoniyatlariga ko‘ra bir nechta guruhga ajratish qabul qilingan.

a) superkompyuterlar;

b) shaxsiy kompyuterlar.

Superkompyuterlar murakkab tuzilmaga ega bo‘lib, o‘ta katta hajmdagi ishlarni (sondagi amallarni) bajara oladi. Masalan, Titan superkompyuteri 18688 ta protsessordan tashkil topgan va 405 kv. m maydonni egallaydi. Amallarni bajarish tezligi 20 petaflopsni (1 petaflops – sekundiga 10^{15} ta amal) tashkil qiladi. Operativ xotirasi esa 710 terrabayt.

Shaxsiy kompyuterlardan joriy vaqtda faqat bir kishi foydalana oladi. Bunday kompyuterlar statsionar va mobil (noutbuk, netbuk, planshet) turlarga ajratiladi.

Statsionar kompyuterlar odatda bitta stol ustida joylashadi va turli murakkablikdagi masalalarni hal qilishga mo‘ljallanadi. Qurilmalari bir nechta korpuslarda saqlanadi.

Noutbuklar – bloknot kompyuterlar statsionar kompyuterlarga qaraganda imkoniyatlari pastroq hisoblanadi va asosiy qurilmalari bitta korpusga joylashtiriladi. Foydalanuvchi bunday kompyuterlarni o‘zi bilan olib yura oladi.

Titan superkompyuteri (2012-y.)

Statsionar kompyuter

Noutbuk

Netbuk

Planshet

Netbuklar asosan internet bilan ishlash yoki unchalik murakkab bo‘lmagan masalalarni bajarishga mo‘ljallangan bo‘lib, noutbuklarning soddalashtirilgan ko‘rinishini ifodalaydi.

Planshetlar boshqa mobil kompyuterlardan barcha qurilmalarning bitta korpusga joylashtirilganligi hamda sezgir (sensorli) ekrani bilan farqlanadi. Ularning imkoniyatlari noutbuk va netbuklarga qaraganda ancha past.

Qanday turda bo‘lishidan qat’iy nazar, kompyuterlar quyidagi qurilmalardan tashkil topadi:

1) asosiy qurilmalar (tizimli blok, kiritish va chiqarish qurilmalari, xotira, protsessor);

2) qo‘shimcha qurilmalar.

Kompyuterning asosiy qurilmalarining birortasi mavjud bo‘lmasa yoki nosoz bo‘lsa, kompyuter umuman ishlamaydi.

Tizimli blok o‘z ichiga ona (o‘zak) platasi, ta’minot qurilmasi hamda sovutgichlarni oladi.

Ona plata o‘zida kompyuterning barcha qurilmalarini birlashtiradi. U orqali kompyuterning barcha qurilmalari “o‘zaro muloqot”ga kirishadi. Bu plata shinalari (qurilmalarni bog‘lovchi kanallar) qanchalik tez ishlasa, kompyuterning tezligi shunchalik katta bo‘ladi.

Ta’minot qurilmasi kompyuterning barcha qurilmalarini elektr energiyasi bilan ta’minlaydi.

Sovutgichlar (kulerlar) elektr energiyasi orqali ishlab, qizib ketishi mumkin bo‘lgan qurilmalarni sovutadi.

Kompyuterning kiritish-chiqarish qurilmasi o‘z ichiga monitor, klaviatura, sichqonchani oladi.

Monitor - kompyuter joriy vaqtda nima bilan mashg‘ul bo‘layotganligini kuzatish, kiritilayotgan va chiqarilayotgan ma’lumotlarni nazorat qilish uchun mo‘ljallangan.

Sichqoncha kompyuter ko‘rsatkichini monitor bo‘ylab boshqarishni osonlashtiradi. Ko‘plab adabiyotlarda ular yordamchi qurilma sifatida tavsiflanadi. Ammo “sichqoncha” allaqachon asosiy qurilmaga aylangan va zamonaviy kompyuterlarda bu qurilmasiz ishlashni tasavvur qilish ham mumkin emas.

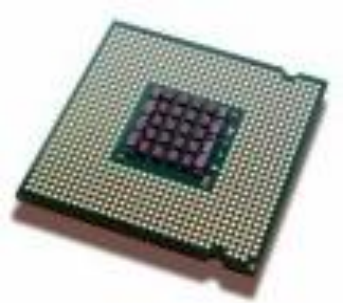
Klaviatura esa kompyuter xotirasiga turli xarakterdagi ma’lumotlarni (matnlar, raqamlar, tinish belgilari, buyruqlar va h.k) kiritish, kompyuterga ko‘rsatmalar berish uchun foydalaniladi.

Monitor

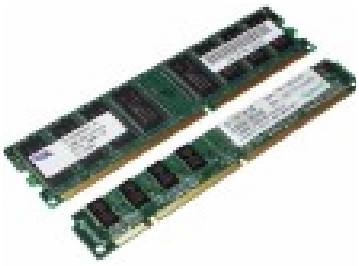
Klaviatura va sichqoncha

Ona plata

Protsessor – kompyuterning “miyasi” hisoblanadi. U kompyuterdagi barcha arifmetik va mantiqiy amallarni bajaradi. Kompyuter ishlash tezligini belgilashda protsessor chastotasi (amallarni bajarish tezligi) muhim ahamiyat kasb etadi. Kompyuterning markasi va imkoniyatlari unga o‘rnatilgan protsessor nomi va soniga ko‘ra baholanadi.



Protsessor



Operativ xotira

Vinchester

Xotira ma’lumotlarni saqlash qurilmasi sanaladi va ikki turga bo‘linadi: doimiy va operativ xotira. Doimiy xotira odatda vinchester deb yuritiladi va ma’lumotlarni uzoq muddatga saqlash imkonini beradi. Vinchesterda ma’lumotlarni saqlash yoki o‘chirish kompyuter foydalanuvchisi tomonidan boshqariladi.

Operativ xotira (tezkor xotira) joriy vaqtda kompyuter yordamida yechilayotgan masala uchun zarur bo‘lgan ma’lumotlarni saqlash qurilmasi hisoblanadi. Vaqt o‘tishi bilan masala o‘zgaradi va yangi masala uchun o‘zga ma’lumotlar talab qilinadi. Foydalanib bo‘lgan ma’lumotlar doimiy xotiraga qaytariladi va ular o‘rniga yangi ma’lumotlar chaqiriladi. Shuning uchun doimiy va operativ xotiralar o‘rtasida muntazam ravishda ma’lumotlar almashib turadi. Kompyuterning ishlash tezligi bevosita operativ xotira hajmiga bog‘liq bo‘ladi.

3.3. Kompyuterning yordamchi qurilmalari

Imkoniyatlaridan yanada to‘laroq foydalanish maqsadida zamonaviy kompyuterlar tarkibiga bir qator yordamchi qurilmalarni qo‘shish mumkin.

Videokarta tasvirlarni monitorga uzatish uchun tayyorlaydi. Tasvirlarning sifati va monitordagi tezligi videokartaga bog‘liq. U o‘z ichki operativ xotirasi va protsessoriga ega. Videokarta protsessorining chastotasi va xotirasining hajmi monitordagi tasvirlarning sifati va tezligiga to‘g‘ridan-to‘g‘ri ta’sir ko‘rsatadi.

Tovush kartasi tovushli ma’lumotlarni tayyorlash va kolonkalarga uzatish vazifasini bajaradi. Odatda tovush kartasi ona plata tarkibiga kiritilgan bo‘ladi. Ammo ehtiyoj bo‘lsa, uni alohida ham qo‘shish mumkin.

Tarmoq kartasi – kompyuterni boshqa kompyuterlar bilan mahalliy yoki global tarmoqlar (masalan, internet) orqali aloqa o'rnatish imkonini beradi. U ham ona platasiga o'rnatilgan yoki alohida joylashtirilgan bo'lishi mumkin.

Disk yuritgich − CD/DVD (ROM yoki REWRITER) qurilmasi CD va DVD tipidagi kompakt disklarga yozish va ulardagi ma'lumotlarni o'qish uchun mo'ljallangan. Bu qurilmalar bir-biridan ma'lumotlarni o'qish va yozish tezligi bilan farqlanadi. CD disklariga 720 Mbayt, DVD disklarga esa 4,2 Gbaytgacha ma'lumot yozish mumkin.

Videokarta

Tovush kartasi

Tarmoq kartasi

Printerlar kompyuter xotirasidagi ma'lumotlarni qog'ozga bosib chiqarish uchun mo'ljallangan. Bugungi kunda lazer kukuni yoki siyoh bilan ishlaydigan printerlardan keng foydalaniladi. Odatda, siyohli printerlar maxsus katrijlar bilan ta'minlangan bo'lib, rangli matn va tasvirlarni bosib chiqarishda qo'llanadi. Bunday printerlarning kamchiligi shuki, ulardan foydalanish ancha qimmat turadi va lazer printerlarga nisbatan tezligi kam bo'ladi.

Lazer kukuni bilan ishlaydigan printerlar oq-qora yoki rangli bo'ladi. Odatda ular minutiga 15-20 tagacha qog'ozga matn bosib chiqarishi mumkin.

Skaner hujjatlar, fotosurat va grafik tasvirlarni kompyuter xotirasiga ko'chirish qurilmasi bo'lib, bir-biridan ishlash tezligi va sifati bilan farqlanadi.

Dinamiklar kompyuter xotirasidagi tovushli ma'lumotlar, musiqa va maxsus signallarni eshitish uchun mo'ljallangan.

Bu qurilmalardan tashqari, kompyuterlarga djoystik, rul va pedal kabi qurilmalarni ham o'rnatish mumkin.

Takrorlash uchun savol va topshiriqlar

1. Kompyuterdan foydalanuvchilarni qanday guruhlarga ajratish mumkin?
2. Kompyuterlar qanday guruhlarga ajratiladi va ular bir-biridan nimasi bilan farq qiladi?
3. Kompyuterning asosiy qurilmalari nimalardan iborat va ularning har birining vazifasini ayting.
4. Operativ xotira nima va u qanday vazifani bajaradi?
5. Qo‘shimcha qurilmalar va ularning vazifalari nimalardan iborat?

4-§. HISOBLASH SISTEMALARINING ARIFMETIK ASOSLARI

4.1. Sanoq sistemalari haqida umumiy tushunchalar

Ixtiyoriy sonlarni ifodalashda chekli sondagi raqamlardan foydalanish usuli ***sanoq sistemasi*** deyiladi. Raqamlarning umumiy soni sanoq sistemasining asosi deb ataladi.

Biz kundalik amaliyotda asosan o‘nli sanoq sistemasi bilan ish olib boramiz. Mazkur sistemada o‘nta raqam mavjud (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) bo‘lib, ular yordamida ixtiyoriy sonni ifodalash mumkin.

O‘nlik sanoq sistemasi Al-Xorazmiy tomonidan taklif etilgan bo‘lib, boshqa sanoq sistemalaridan ilmiy hisoblash ishlarida qulayligi va to‘rtta amalni bajarish qoidalarining ishlab chiqilganligi bilan farq qiladi.

Umuman aytganda, p asosli sanoq sistemasida barcha sonlarni ifodalashda 0 dan $p-1$ gacha bo‘lgan natural sonlardan foydalaniladi. Masalan, 2 lik sanoq sistemasida faqat ikkita raqam, ya’ni 0 va 1 raqamlari, 7 lik sanoq sistemasida esa 0 dan 6 gacha raqamlar mavjud.

O‘nli bo‘lmagan asosdagi sanoq sistemalari bilan ishlaganda, ularning asosini indeks tarzida ko‘rsatib qo‘yish qabul qilingan. Masalan, $1101_2, 123_5$ va h.k.

Asosi 10 dan katta bo‘lgan sanoq sistemalarida raqamlar sifatida maxsus belgilardan foydalaniladi. Masalan, 16 lik sanoq sistemasida 10 va undan katta bo‘lgan raqamlarni ifodalash uchun $10=\overline{0}_{16}$; $11=\overline{1}_{16}$; $12=\overline{2}_{16}$; $13=\overline{3}_{16}$; $14=\overline{4}_{16}$; $15=\overline{5}_{16}$ yoki 10=A, 11=B, 12=C, 13=D, 14=E, 15=F belgilashlar qabul qilingan.

4.2. Sonlarni o‘nli sanoq sistemasiga o‘tkazish

Faraz qilaylik, qandaydir son p asosli sanoq sistemasida n ta $a_i, i=1,....,n$ raqamlar yordamida berilgan bo‘lsin. U o‘nli sanoq sistemasida qanday songa teng bo‘ladi? Bu savolga quyidagi formula javob beradi:

$$a_1a_2...a_n{}_p = a_1\cdot p^{n-1}+a_2\cdot p^{n-2}+...+a_{n-1}\cdot p+a_n. \qquad (4.1)$$

1-misol. 7 lik sanoq sistemasida berilgan 23651_7 sonini o‘nlik sanoq sistemasida ifodalang.

Yechish. Berilgan sonning 5 ta raqamdan iborat ekanligini e’tiborga olib, (4.1) formulani qo‘llaymiz:

$$23651_7 = 2\cdot 7^4+3\cdot 7^3+6\cdot 7^2+5\cdot 7+1=2\cdot 2401+3\cdot 147+$$

$$+6\cdot 49+5\cdot 7+1=4802+441+294+35+1=5573.$$

Demak, $23651_7=5573_{10}$.

2-misol. 16 lik sanoq sistemasida berilgan $2AB_{16}$ sonini o‘nlik sanoq sistemasida ifodalang.

Yechish. Berilgan sonning 3 ta raqamdan iborat ekanligi hamda 16 lik sanoq sistemasida $A=10$, $B=11$ ekanligini e’tiborga olib, (4.1) formulani qo‘llaymiz:

$$2AB_{16}=2\cdot 16^2+10\cdot 16+11=2\cdot 256+160+11=783.$$

Demak, $2AB_{16}=783_{10}$.

4.3. Sonlarni boshqa sanoq sistemasiga o‘tkazish

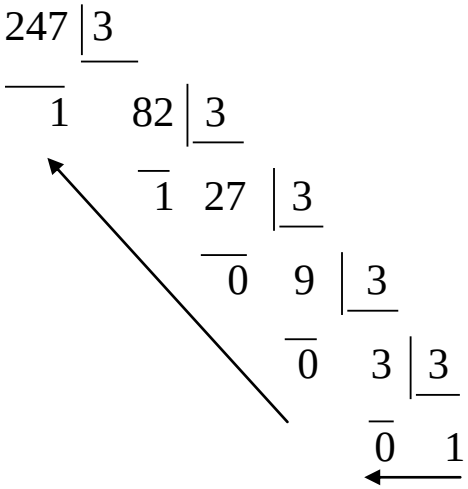
Butun sonlarni o‘tkazish. Aytaylik, 10 lik sanoq sistemasidagi N sonini q lik sanoq sistemasiga o‘tkazish talab qilinsin.

O‘tkazish quyidagi qoidaga binoan bajariladi. N soni va uning qoldiqlarini q ga ketma-ket bo‘lish amali orqali bajariladi. Bo‘lish amali to qoldiqda q dan kichik son paydo bo‘lmaguncha davom

ettiriladi. So‘ngra hosil qilingan qoldiqlar teskari tartibda yozib chiqiladi.

3-misol. 247_{10} sonini 3 lik sanoq sistemasiga o‘tkazing.

Yechish:



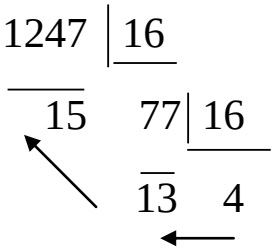
Demak, $247_{10} = 100011_3$

Tekshirib ko‘ramiz.

$$100011_3 = 1\cdot 3^5 + 0\cdot 3^4 + 0\cdot 3^3 + 0\cdot 3^2 + 1\cdot 3 + 1 =$$
$$= 243 + 0 + 0 + 0 + 3 + 1 = 247$$

4-misol. 1247_{10} sonini 16 lik sanoq sistemasiga o‘tkazing.

Yechish:



Demak, $1247_{10} = 4DF_{16}$. Natijani tekshirish o‘zingizga havola.

Haqiqiy sonlarni boshqa sistemaga o‘tkazish. Aytaylik, o‘nlik sanoq sistemasidagi haqiqiy sonni *q*-lik sanoq sistemasiga o‘tkazish talab qilinsin.

Berilgan haqiqiy sonni *q*-lik sanoq sistemasiga o‘tkazishda uning butun qismini alohida, kasr qismini alohida yangi sistemaga o‘tkaziladi. Sonning butun qismini boshqa sanoq sistemasiga o‘tkazishni biz yuqorida ko‘rdik. Uning kasr qismini *q*-lik sanoq sistemasiga o‘tkazish uchun to‘g‘ri kasr va hosil bo‘lgan ko‘paytmalarning kasr qismlarini *q* ga ketma-ket ko‘paytirish yo‘li bilan amalga oshiriladi. *Q*-lik

sanoq sistemasiga raqamlarni topish uchun har galgi ko‘paytmaning butun qismi ajratib olinadi.

5-misol. O‘nlik sanoq sistemasidagi 0,41 sonini ikkilik sanoq sistemasiga o‘tkazing. (r=10, q=2):

Yechish:

$$\begin{array}{r} 0,41 \\ \times\ \underline{2} \\ 0,82 \\ \times\ \underline{2} \\ 1,64 \\ \times\ \underline{2} \\ 1,28 \\ \times\ \underline{2} \\ 0,56 \\ \times\ \underline{2} \\ 1,12 \\ \times\ \underline{2} \\ 0,24 \\ \times\ \underline{2} \\ 0,48 \end{array}$$

Demak, $0,41_{10} = 0,0110100_2$

Bu misoldan ko‘rinib turibdiki, kasr sonlarni boshqa sanoq sistemasiga o‘tkazish umumiy holda cheksiz jarayon bo‘lishi ham mumkin. Bunday masalalar taqribiy usulda amalga oshiriladi, ya’ni aniqlik belgilab beriladi va ko‘paytirishlar soni ana shu aniqlikka teng bo‘ladi.

Eslatma. Ko‘paytirish amali natijaviy sonning faqat kasr qismiga nisbatan qo‘llanadi.

6-misol. $118,375_{10}$ sonini 4 lik sanoq sistemasiga o‘tkazing.

Yechish. Berilgan sonning butun va kasr qismlarini ajratib olamiz:

$$\begin{array}{r|l} 118 & 4 \\ \hline 2 & 29 \\ \hline 1 & 7 \\ \hline 3 & 1 \end{array} \qquad \begin{array}{r} 0,375 \\ *\ \underline{4} \\ 1,500 \\ *\ \underline{4} \\ 2,000 \end{array}$$

Demak, $118,375_{10} = 118_{10} + 0,375_{10} = 1312_4 + 0,12_4 = 1312,12_4$.

4.4. Ikkilik-sakkizlik va ikkilik-o‘n oltilik sanoq sistemalari

Kompyuter xotirasida hamma ma’lumotlar ikkilik sanoq sistemasida ifodalanadi. Ammo bu degani barcha sonli ma’lumotlar biz yuqorida keltirgan usulda ikkilik sanoq sistemasiga o‘tkazilishini anglatmaydi. Ikkilik-sakkizlik va ikkilik-o‘n oltilik sanoq sistemalarini qo‘llash natijasida 2 ga bo‘lish amallari sonini qisqartirishga erishish mumkin.

Ikkilik-sakkizlik sanoq sistemasida sakkizlik sanoq sistemasidagi 0 dan 7 gacha bo‘lgan raqamlar uchta nol va birlar orqali ifodalanadi. To‘g‘rirog‘i, bu raqam ikkilik sanoq sistemasiga o‘tkaziladi va chap tomondan 0 lar hisobiga uchtagacha to‘ldiriladi. Bunda quyidagi jadvaldan foydalaniladi:

Sakkizlik	Ikkilik-sakkizlik	Sakkizlik	Ikkilik-sakkizlik
0	000	4	100
1	001	5	101
2	010	6	110
3	011	7	111

O‘nlik sanoq sistemasida berilgan har qanday son avval sakkizlik sanoq sistemasiga o‘tkaziladi. Shunday keyin natijaviy sondagi har bir raqam yuqoridagi jadval asosida ikkilik-sakkizlik sistemadagi raqamlar uchligi bilan almashtiriladi.

7-misol. 123_{10} soni ikkilik-sakkizlik sanoq sistemasida yozilsin.

Yechish. Dastlab 123 sonini 8 lik sanoq sistemasiga o‘tkaziladi. $123_{10} = 173_8$. Endi hosil qilingan 173_8 ning har bir raqami yuqoridagi jadval yordamida almashtiriladi:

$$173_8 = \underset{1}{0}\underset{7}{01}\,\underset{3}{111}\,011$$

Demak, 123_{10} soni ikkilik-sakkizlik sanoq sistemasida 001 111 011 tarzida yoziladi.

Ikkilik-o‘n oltilik sanoq sistemasida 0 dan 15 gacha bo‘lgan sonlar to‘rtta nol va birlar orqali ifodalanadi. To‘g‘rirog‘i, bu raqam

ikkilik sanoq sistemasiga o‘tkaziladi va chap tomondan 0 lar hisobiga to‘rttagacha to‘ldiriladi. Bunda quyidagi jadvaldan foydalaniladi.

O‘n oltilik	Ikkilik-o‘n oltilik	O‘n oltilik	Ikkilik-o‘n oltilik
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

O‘nlik sanoq sistemasida berilgan har qanday son avval o‘n oltilik sanoq sistemasiga o‘tkaziladi. Shunday keyin natijaviy sondagi har bir raqam yuqoridagi jadval asosida ikkilik-o‘n oltilik sistemadagi raqamlar to‘rtligi bilan almashtiriladi.

8-misol. 1234_{10} sonini ikkilik-o‘n oltilik sanoq sistemasiga o‘tkazing.

Yechish. Dastlab 1234_{10} sonini o‘n oltilik sanoq sistemasiga o‘tkaziladi: $1234_{10} = 4D2_{16}$. Shundan keyin $4D2_{16}$ sonidagi raqamlarni mos ravishda ikkilik-o‘n oltilik sonlar jadvali yordamida almashtiriladi:

$$4D2_{16} = \underbrace{0100}_{4}\ \underbrace{1101}_{D}\ \underbrace{0010}_{2}.$$

Demak, $4D2_{16}$ soni ikkilik-o‘n oltilik sanoq sistemasida 0100 1101 0010 tarzida yozilar ekan.

Takrorlash uchun savol va topshiriqlar

1. Sanoq sistemasi nima?
2. 5 lik sanoq sistemasidagi raqamlarni ayting.
3. Ikkilik sanoq sistemasida asosiy arifmetik amallarning bajarilishini misollar orqali tushuntiring.

4. Nima uchun hisoblash mashinalarida ikkilik-sakkizlik va ikkilik-o‘n oltilik sanoq sistemalaridan foydalaniladi?

5. Quyidagi misollarni bajaring:

a) $100001001110_2 \rightarrow X_{10}$; d) $124_{10} \rightarrow X_2$;

b) $105_{10} \rightarrow X_8$; e) $645_8 \rightarrow X_{10}$;

c) $(0,54)_{10} \rightarrow X_2$; f) $(405,5)_8 \rightarrow X_2$;

6. O‘nlik sanoq sistemasida berilgan quyidagi sonlarni ikkilik-sakkizlik va ikkilik-o‘n oltilik sanoq sistemasida ifodalang:

a) 1234; c) -156356;

b) 4536,25; d) -23895,125;

5-§. MATEMATIK MANTIQ ELEMENTLARI

5.1. Mulohaza tushunchasi

Mulohazalar algebrasi hamda matematik mantiq elementlari informatikada o‘ta muhim ahamiyat kasb etadi. Ular informatikaning barcha bo‘lim va masalalarida ikkilik sanoq sistemasidagi mantiq, algoritm va dasturlarning ishlash qoidalari, kompyuterlarning ishlash prinsipi, dasturiy ta’minot ishlab chiqish kabi ko‘rinishlarda ishtirok etadi.

Shunday qilib, informatika uchun mulohazalar algebrasi ajralmas va fundamental tushuncha sanaladi.

Mulohaza tushunchasi matematik mantiqning boshlang‘ich tushunchalaridan biri hisoblanadi.

Matematik mantiqda mulohaza deb, rost yoki yolg‘onligi haqida xulosa chiqarish mumkin bo‘lgan har qanday darak gapga aytiladi.

Masalan, “Bugun 7-sentabr 2013-yil”, “Hozir yomg‘ir yog‘-yapti”, “12 ni 3 ga bo‘lsa 4 bo‘ladi” va h.k.

1. So‘roq va undov gaplar mulohaza hisoblanmaydi. Masalan, “Soli maktabga kelasanmi?”, “Yashasin yangi yil!”.

2. Ba’zi jumlalarning rost yoki yolg‘onligini bir qiymatli aniqlash mumkin emas. Masalan: “Lolaxon chiroyli qiz”, “Romsozlik qiyin kasb”, “Bugun havo yaxshi emas”.

3. Shunday jumlalar ham borki, ularning tarkibida umumiy nomlar − noma'lumlar ishtirok etadi. Masalan, "x son y songa bo'linadi", "$\sin x + \cos x = 0$", "x Norin tumanining markazi", "x qiz y qizning dugonasi". Biz bu gaplarni mulohaza deya olmaymiz, chunki ularning rost yoki yolg'onligi noma'lumlarning qiymatiga bog'liq bo'lib qoladi. Masalan, birinchi jumla $x = 20,\ y = 5$ uchun rost, ammo $x = 24,\ y = 7$ uchun yolg'on, 4-jumla esa x va y lar o'rniga qizlar ismlarini qo'yilganda mulohazalarga aylanishi mumkin.

3-qabilida hosil qilingan barcha mulohazalar (jumla) predikatlar deb ataladi.

Mulohazalarni katta lotin harflari (A, B, C, ...) bilan belgilash qabul qilingan.

Agar mulohazani ikkita mulohazaga ajratish mumkin bo'lmasa, bunday mulohazalarni elementar (sodda) deyiladi. Masalan, "6 − tub son".

Agar A va B mulohazalar berilgan bo'lsa, ulardan "va", "yoki", "agar ... bo'lsa, u holda ... bo'ladi" hamda "emas" bog'lovchilari yordamida murakkab mulohazalarni hosil qilish mumkin.

"Agar to'rtburchakning qarama-qarshi tomonlari teng bo'lsa, u holda bu to'rtburchak parallelogrammdir", "Agar 2>3 bo'lsa, u holda London shahri Germaniyaning poytaxtidir" mulohazalari esa murakkab mulohazalardir.

Mantiqda mulohazalar ma'nosi e'tiborga olinmaydi, faqat ularning rost yoki yolg'onligi bilan ish olib boriladi.

Agar A elementar mulohaza rost bo'lsa, uning rostlik qiymati – 1 ("rost"), aks holda – 0 ("yolg'on") qiymatini qabul qiladi.

Murakkab mulohazalarning rostlik qiymati tarkibidagi elementar mulohazalarning rostlik qiymatlaridan foydalanib aniqlanadi.

5.2. Mulohazalarning inkori, dizyunksiyasi, konyunksiyasi, implikatsiyasi va ekvivalensiyasi

1-ta'rif. A mulohaza rost bo'lganda yolg'on, yolg'on bo'lganda rost bo'ladigan mulohazaga A ning inkori deyiladi.

A mulohazaning inkorini $\overline{A}$ ko‘rinishida belgilanadi va “A emas” deb o‘qiladi.

A mulohaza “3 soni 432 ning bo‘luvchisi” bo‘lsa, u holda $\overline{A}$ mulohaza “3 soni 432 ning bo‘luvchisi emas” bo‘ladi.

A va B elementar mulohazalarni “yoki” bog‘lovchisi bilan birlashtirib, A va B mulohazalarning dizyunksiyasi deb ataladigan yangi mulohazaga ega bo‘lamiz.

2-ta’rif. A va B mulohazalardan kamida bittasi rost bo‘lgan holda rost qiymatini oladigan mulohazalarga A va B larning dizyunksiyasi deyiladi. Dizyunksiya $A \vee B$ ko‘rinishda belgilanadi.

Masalan, “21>3” va “$21=6$” mulohazalardan “21>3 yoki $21=6$” dizyunksiyasini hosil qilaylik. Ko‘rish qiyin emaski, u tarkibidagi “21>3” elementar mulohaza rost bo’lgani uchun rost qiymatini oladi.

“Quyosh oy atrofida aylanadi yoki 24 soni 0 ga bo‘linadi” dizyunksiyasi yolg‘on, chunki uni tashkil etuvchi mulohazalarning har ikkisi ham yolg‘on.

A va B elementar mulohazalarni “va” bog‘lovchisi orqali birlashtirib, A va B mulohazalarning konyunksiyasi deb ataladigan yangi mulohazaga ega bo‘lamiz.

3-ta’rif. A va B mulohazalar bir vaqtda rost bo‘lganda rost, boshqa hollarda yolg‘on qiymatini oladigan mulohazaga A va B mulohazalarning konyunksiyasi deyiladi va $A \wedge B$ ko‘rinishida belgilanadi.

Mulohazalar konyunksiyasini mantiqiy ko‘paytma ham deb ham yuritiladi.

“2×2=4 va 16>10” konyunksiyasi - rost, “12>20 va London Angliyaning poytaxti” esa – yolg’on.

4-ta’rif. Faqat A mulohaza rost, B - yolg‘on bo‘lgandagina yolg‘on qiymatini oladigan mulohaza A va B mulohazalarning implikatsiyasi deyiladi va u $A \Rightarrow B$ ko‘rinishida belgilanadi.

A va B mulohazalar implikatsiyasi “agar A bo‘lsa, u holda B bo‘ladi” yoki “A mulohazadan B mulohaza kelib chiqadi” deb

o‘qiladi. *A* ni implikatsiyaning sharti, *B* ni esa uning xulosasi deb atash qabul qilingan.

A mulohaza “Kecha yakshanba edi”, *B* mulohaza esa “Men ishda bo‘lmadim” bo‘lsin. Bu vaziyatda “Agar kecha yakshanba bo‘lsa, u holda men ishda bo‘lmadim” murakkab mulohazaning shakli “Agar *A* bo‘lsa, u holda *B* bo‘ladi” ko‘rinishiga ega bo‘ladi.

5-ta’rif. Faqat *A* va *B* mulohazalar bir xil qiymat qabul qilganda rost bo‘lib, qolgan barcha hollarda yolg‘on bo‘ladigan mulohazaga *A* va *B* mulohazalarning ekvivalensiyasi deyiladi va $A \Leftrightarrow B$ ko‘rinishida belgilanadi.

A va *B* mulohazalarning ekvivalensiyasini “*A* ekvivalent *B*” yoki “*A* mulohazalarning bajarilishi uchun *B* mulohazaning bajarilishi zarur va yetarlidir”, “*A* bajarilgan holda va faqat shu holda *B* bajariladi” deb o‘qiladi.

A va *B* mulohazalar inkori, dizyunksiya, konyunksiya, implikatsiya va ekvivalensiyasining rostlik jadvali quyidagicha ko‘rinishda bo‘ladi:

A	B	$\overline{A}$	$A \vee B$	$A \wedge B$	$A \Rightarrow B$	$A \Leftrightarrow B$
0	1	1	1	0	1	0
0	0	1	0	0	1	1
1	1	0	1	1	1	1
1	0	0	1	0	0	0

5.3. Aynan rost, aynan yolg‘on va bajariluvchi mulohazalar

$F(A_1, A_2, ..., A_n)$ mulohaza tarkibiga faqat $A_1, A_2, ..., A_n$ elementar mulohazalar kirgan ixtiyoriy murakkab mulohaza bo‘lsin. Yuqoridan ma’lumki, har bir elementar mulohaza “rost” yoki “yolg‘on” qiymatni qabul qiladi. Murakkab mulohazaga kirgan har bir elementar mulohazani 0 yoki 1 bilan almashtirsak, $A_1, A_2, ..., A_n$ elementar mulohazalarning qiymatlaridan tuzilgan hamda 1 va 0 lardan tashkil top-

gan jamlanma hosil bo‘ladi. Bunday jamlanmani umumiy holda $\alpha=(\alpha_1, \alpha_2, ..., \alpha_n)$ ko‘rinishda belgilaymiz va uni $A_1, A_2, ..., A_n$ elementar mulohazalar qiymatlarining jamlanmasi deymiz.

Ravshanki, bunda $\alpha_i, i=1, ..., n$ belgilar 1 yoki 0 lardan faqat bittasini qabul qiladi. Odatda (1, 1, …, 1) jamlanmani “birinchi”, (0, 0, …, 0) ni esa “oxirgi” jamlanma deyiladi. Yana shuni ham aytish kerakki, bu jamlanmalardagi belgilar soni $F(A_1, A_2, ..., A_n)$ murakkab mulohazadagi elementar mulohazalar soniga teng bo‘ladi. Biz qarayotgan holda belgilarning soni *n* ga (*n*-natural son) teng bo’lib, bu son jamlanmaning uzunligi deyiladi. *N* ta belgidan tashkil topgan ikkita jamlanmani bir-biridan farqlash maqsadida, jamlanmalarning tengligi tushunchasi kiritiladi.

$\alpha=(\alpha_1, \alpha_2, ..., \alpha_n)$ jamlanma $\beta=(\beta_1, \beta_2, ..., \beta_n)$ ga faqat $\alpha_1=\beta_1, ..., \alpha_n=\beta_n$ bo‘lgan holdagina teng deyiladi.

6-ta’rif. Murakkab mulohaza tarkibidagi elementar mulohazalar qabul qilishi mumkin bo‘lgan barcha qiymatlarining jamlanmalarida rost bo‘lsa, u holda aynan rost mulohaza yoki tavtologiya deyiladi.

$A \wedge B \Leftrightarrow B \wedge A$, $A \vee \overline{A}$, $A \Rightarrow A$ mulohazalar tavtologiya ekanligi ravshan.

7-ta’rif. Tarkibidagi barcha elementar mulohazalarning qabul qilishi mumkin bo‘lgan qiymatlari jamlanmalaridan kamida bittasida rost bo‘ladigan murakkab mulohazalar bajariluvchi mulohazalar deb ataladi.

8-ta’rif. Murakkab mulohaza elementar mulohazalarining qabul qilishi mumkin bo‘lgan barcha qiymatlari jamlanmalarida yolg‘on bo‘lsa, bunday mulohazalarni aynan yolg‘on deyiladi.

5.4. Mulohazalar algebrasining formulalari

Agar elementar mulohazalar *A*, *B* kabi o‘zgaruvchilar bilan belgilangan bo‘lsa, ularni propozitsional o‘zgaruvchilar deb ataladi.

Mulohazalar algebrasining asosiy tushunchalaridan biri formula hisoblanadi. Ularni qurish uchun propozitsional o‘zgaruvchilar hamda

$\Leftrightarrow, \Rightarrow, \vee, \wedge,$ inkor kabi mantiqiy amal belgilari hamda chap va o‘ng qavslardan foydalanish mumkin.

Formula tushunchasi quyidagicha ta’riflanadi.

Ta’rif. 1. Har bir propozitsional o‘zgaruvchi - formula.

2. Agar F va G lar formula bo‘lsa, u holda $(F \Leftrightarrow G)$, $(F \Rightarrow G)$, $(F \wedge G)$, $(F \vee G)$, $\overline{F}$ ifodalar ham formulalardir.

3. Mulohazalar algebrasining formulalari faqat 1- va 2-punktlar yordamida hosil qilinadi.

1-misol: $((A \Rightarrow B) \vee F)$ ifoda formuladir.

2- misol: $A \vee B$ ifoda formula emas, chunki bu formulada tashqi qavslar yetishmaydi.

5.5. Teng kuchli formula va almashtirishlar

Mulohazalar algebrasining ixtiyoriy formulasi o‘zining rostlik jadvali bilan xarakterlanadi.

1-misol. $A \Rightarrow B \wedge \overline{C}$ formulaga ushbu rostlik jadvali mos keladi.

A	B	C	$\overline{C}$	$B \wedge \overline{C}$	$A \Rightarrow B \wedge \overline{C}$
1	1	1	0	1	0
1	1	0	1	0	1
1	0	1	0	0	0
1	0	0	1	0	0
0	1	1	0	0	1
0	1	0	1	1	1
0	0	1	0	0	1
0	0	0	1	0	1

Ta’rif. Agar mulohazalar algebrasining $F_1(A_1, ..., A_n)$ va $F_2(A_1, ..., A_n)$ formulalari propozitsional o‘zgaruvchilar mos qiymatlarining barcha jamlanmalarida bir xil qiymat qabul qilsa, bu formulalar teng kuchli formulalar deyiladi.

$F_1(A_1, ..., A_n)$ va $F_2(A_1, ..., A_n)$ formulalarning teng kuchli ekanligi $F_1(A_1, ..., A_n) \equiv F_2(A_1, ..., A_n)$ ko‘rinishda yoziladi.

Ta’rifga ko‘ra 1- va 2-misollardagi formulalar teng kuchlidir, ya’ni $A \Rightarrow B \wedge \overline{C} \equiv (\overline{A} \vee B) \wedge \overline{(A \wedge C)}$.

Mantiqiy amallar ta’rifidan foydalanib ayrim tengkuchliliklarni bevosita isbotlash mumkin, masalan:

$$A \wedge B \equiv B \wedge A,\ \overline{\overline{A}} \equiv A,\ \overline{A} \wedge A \equiv 0,\ \overline{A} \vee A \equiv 1$$

munosabatlar o‘rinlidir.

Ta’rifga ko‘ra, formulalarning teng kuchli ekanligini aniqlashning umumiy usuli quyidagicha: har bir formula uchun rostlik jadvali tuziladi, propozitsional o‘zgaruvchilarning bir xil jamlanmalari uchun formulalar qabul qiladigan qiymatlari solishtiriladi. Agar jamlanmalarning barcha mos kombinatsiyalarida formulalar bir xil qiymatga ega bo‘lsa, bu formulalar teng kuchli bo‘ladi.

Quyidagi formulalar mulohazalar algebrasining asosiy teng kuchliliklari hisoblanadilar:

1. $A \wedge B \equiv B \wedge A$ (konyunksiyaning o‘rin almashuvchanligi).
2. $A \vee B \equiv B \vee A$ (dizyunksiyaning o‘rin almashuvchanligi).
3. $A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C$ (konyunksiyaning assotsiativligi).
4. $A \vee (B \vee C) \equiv (A \vee B) \vee C$ (dizyunksiyaning assotsiativligi).
5. $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$ (dizyunksiyaning konyunksiyaga nisbatan distributivligi).
6. $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$ (konyunksiyaning dizyunksiyaga nisbatan distributivligi).
7. $A \wedge A \equiv A$ (konyunksiyaning idempotentligi).
8. $A \vee A \equiv A$ (dizyunksiyaning idempotentligi).
9. $A \wedge 1 \equiv 1$. 10. $A \vee 1 \equiv 1$. 11. $A \wedge 0 \equiv A$.
12. $A \vee 0 \equiv A$. 13. $A \wedge \overline{A} = 0$. 14 $A \vee \overline{A} \equiv 1$.
15. $\overline{(A \wedge B)} \equiv \overline{A} \vee \overline{B}$ (de Morgan tengkuchliliklari).
16. $\overline{(A \vee B)} \equiv \overline{A} \wedge \overline{B}$ (de Morgan tengkuchliliklari).

17. $A \Rightarrow B \equiv \overline{A} \vee B$ (implikatsiyaning inkor va konyunksiya bilan ifodalanishi).

18. $\overline{\overline{A}} \equiv A$ (qo‘sh inkor tengkuchliligi).

19. $A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$.

Keltirilgan har bir teng kuchliliklarning o‘rinli ekanligini rostlik jadvallari yordamida isbotlash mumkin. Bu teng kuchliliklar yordamida berilgan formulaga teng kuchli formulalarni hosil qilish, teng kuchliligini aniqlash, soddaroq ko‘rinishga keltirish hamda aynan rost, aynan yolg‘on, bajariluvchan ekanligini aniqlash mumkin.

Har qanday mulohazaga biror formula mos kelishi va formulalar uchun ko‘rib chiqilgan teng kuchliliklarni hisobga olib, mulohazalar uchun ham teng kuchliliklarni qo‘llash mumkin, ya’ni mulohazani boshqa biror teng kuchli mulohazaga almashtirish, berilgan mulohazalarni teng kuchliligini aniqlash, murakkab mulohazani aynan rost yoki aynan yolg‘on ekanligini aniqlash mumkin.

Takrorlash uchun savol va topshiriqlar

1. Mulohaza nima va qanday qiymatlarni qabul qiladi?
2. Mulohazalarning qanday turlari mavjud?
3. Mulohazalar inkori, dizyunksiyasi, konyunksiyasi deganda nimani tushunasiz?
4. Mulohazalarning implikatsiyasi va ekvivalensiyasi nima?
5. Formula va teng kuchli formula tushunchalari ta’riflarini eslang. Misollar keltiring.

6-§. KOMPYUTER XOTIRASIDA MA’LUMOTLARNING IFODALANISHI

6.1. Umumiy tushunchalar

Ma’lumot – bu maqsadga yo‘naltirilgan faoliyatni tashkil etish uchun borliq olam va unda sodir bo‘layotgan voqealar haqida inson yoki biror qurilma tomonidan olingan axborotdir.

Kompyuterda qayta ishlashga mo‘ljallangan ma’lumotlar tabiatiga ko‘ra son, matn, grafik, tovush yoki video ko‘rinishlarida bo‘lishi mumkin. Ma’lumotlar o‘zgaruvchan, o‘zgarmas yoki tasodifiy xarakterga ega bo‘lib, ular orasida o‘zgaruvchan ma’lumotlar alohida ahamiyat kasb etadi. Chunki, o‘zgaruvchan ma’lumotlar jarayon va voqealarning sabab va oqibatlarini o‘rganishga imkon beradi.

Har qanday ma’lumot kompyuter xotirasida ikkilik sanoq sistemasida (mashina tilida) 0 yoki 1 lar orqali kodlanadi va saqlanadi (koddagi 0 va 1 lar soniga teng hajmni egallaydi). Kodlash usullari ma’lumotlar ko‘rinishiga bog’liq ravishda bir-biridan farqlanadi.

Mashina tilidagi bitta 0 yoki 1 raqami bir bit ma’lumot deb ataladi. U eng kichik ma’lumot o‘lchovi sanaladi. Undan tashqari ma’lumotlarning quyidagi o‘lchov birliklari mavjud:

Nomi	**Miqdori**
1 bayt	8 bit
1 mashina so‘zi	2 bayt
1 kilobayt (Kbayt)	1024 bayt
1 megabayt (Mbayt)	1024 Kbayt
1 gigabayt (Gbayt)	1024 Mbayt
1 terabayt (Tbayt)	1024 Gbayt
1 petabayt (Pbayt)	1024 Tbayt

Kompyuterda ma’lumotlar asosan bit va baytlarda ifodalanadi. 0 va 1 lardan iborat kombinatsiya bir xil bo‘lsada, tipiga bog‘liq ravishda turli ma’lumotlarni anglatadi.

Bir baytda ifodalash mumkin bo‘lgan butun sonlar diapazoni 8 ta nol va birlarning kombinatsiyasi, ya’ni 2^8=256 bilan chegaralanadi. Ikki bayt bilan esa 0 dan 2^{16}=65536 gacha butun sonlarni yozish mumkin. Undan katta sonlarni saqlash uchun 4 bayt (32 bit) talab qilinadi. Muayyan bitning qiymati uning razryadi (turgan o‘rni) bilan belgilanadi.

Bir mashina so‘zi, ya’ni ikki bayt ma’lumotning razryadlari (bitlari) chapdan o‘ng tomonga qarab nomerlanadi:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

6.2. Sonli ma’lumotlar

Kompyuter xotirasida sonlarni ikki xil o‘lchamda ifodalash mumkin. Ularning birinchisi butun sonlar, ikkinchisi esa haqiqiy sonlar uchun qo‘llanadi.

Kompyuter xotirasida ifodalash mumkin bo‘lgan sonlarning diapazoni chegaralangan bo’lib, bu chegara sonlarni saqlash uchun mo‘ljallangan xotira hajmi bilan belgilanadi. *K*-razryadli yacheykaga 2*K* xil butun sonlarni joylash mumkin.

Butun *N* sonini kompyuter xotirasidagi ko‘rinishini hosil qilish uchun quyidagi amallarni bajarish lozim:

1) *N* sonini ikkilik sanoq sistemasiga o‘tkazish;

2) natijani chap tomondan *K* xonagacha nollar bilan to‘ldirish.

Masalan, 1607 sonini 2 baytli xotiraga joylash talab qilingan bo‘lsin. Buning uchun uni ikkilik sanoq sistemasiga o‘tkaziladi: $1607_{10}=11001000111_2$. Endi hosil qilingan bu sonni chap tomondan 16 ta xonagacha nollar bilan to‘ldiriladi: 0000 0110 0100 0111.

Manfiy butun sonlarni (−*N*) kompyuter xotirasidagi ifodasini topish uchun quyidagi amallarni bajarish lozim:

1) musbat *N* sonining xotiradagi ko‘rinishi aniqlanadi;

2) natija teskari ko‘rinishi yoziladi (0 ni 1, 1 ni esa 0 bilan almashtiriladi);

3) hosil qilingan songa 1 qo‘shiladi.

Masalan, –1607 ni kompyuter xotirasidagi ko‘rinishini topish uchun yuqoridagi misol natijasidan (0000 0110 0100 0111) foydalanish mumkin. Bu sonni teskarisini yoziladi: 1111 1001 1011 1000. Hosil bo’lgan songa 1 ni qo‘shiladi: 1111 1001 1011 1001. Demak, −1607 soni kompyuter xotirasida ana shunday ko’rinishda saqlanadi.

Kompyuterda haqiqiy sonlarni fiksirlangan, ko‘chuvchi vergulli hamda ikkilik-o‘nlik shakllarda ifodalash mumkin.

Fiksirlangan sonlarning ikkilik sanoq sistemasidagi ko‘rinishida vergulning o‘rni qat’iy belgilab qo‘yiladi.

Agar vergul *n*-razryadli sonning birinchi ishonchli raqamidan avval kelsa, bu son moduli bo‘yicha birdan kichik bo‘ladi. Bunday sonlarning diapazoni $2^{-n} \le |A^2| \le 1-2^{-n}$ bo‘ladi.

Agar vergul *n*-razryadli sonning oxirgi ishonchli raqamidan keyin qo‘yilsa, bu son butun bo‘ladi va uning diapazoni $0 \le |A^2| \le 2^n - 1$ ga teng.

Ikkilik sanoq sistemasidagi manfiy sonning eng yuqori razryadi shu sonning ishorasini ko‘rsatadi. Musbat sonning ishora razryadi 0, manfiy sonniki esa 1 ga teng.

Haqiqiy sonlarni ko‘chuvchi vergul yordamida turli ko‘rinishlarda ifodalash mumkin. Masalan:

$$373=0{,}373\cdot 10^3=3{,}73\cdot 10^2=37{,}3\cdot 10^1.$$

Sonning normal ko‘rinishi deb, uni ko‘chuvchi vergul orqali butun qismi nolga teng, verguldan keyingi birinchi raqami noldan farqli holatda ifodalangan ko‘rinishiga aytiladi.

Ko‘chuvchi vergulli sonlar odatda $R = m \cdot 10^p$ shaklida yoziladi. Bu yerda $0 < m < 1$ bo‘lib, sonning mantissasi, *p* esa uning tartibi deb ataladi.

Sonning mantissasi kompyuter xotirasida butun son shaklida saqlanadi (0 va vergul saqlanmaydi). Masalan, 12.345 soni xotira yacheykasiga 12345 tarzida yoziladi. Bu sonni aslini tiklash uchun uchun uning tartibidan foydalaniladi. Bizning misolimizda bu tartib 2 ga teng.

Haqiqiy sonlar odatda to‘rt bayt, ya’ni 64 bit yordamida ifodalanadi. Bunda razryadlar quyidagicha taqsimlanadi:

Ishora razryadi	Tartib razryadlari	Mantissa razryadlari
63	62...52	51...0

Ko‘rish qiyin emaski, mantissa uchun quyi 52 ta, tartibi uchun 11 ta, ishorasi uchun 1 ta razryad ajratilgan.

Son tartibini xotirada saqlash uchun uning o‘nlik sanoq sistemasidagi tartibiga surilish miqdori qo‘shiladi. Masalan, *double* tipidagi son uchun tartib 11 bitni egallaydi va diapazoni 2^{-1023} dan 2^{1023} gacha bo‘ladi.

Shunday qilib, haqiqiy sonlar kompyuter xotirasida quyidagi algoritm asosida ifodalanadi:

1) son modulini ikkilik sanoq sistemasiga o‘tkaziladi;

2) hosil qilingan son normallashtiriladi, ya’ni $m \cdot 2^p$ ko‘rinishida yoziladi. Bu yerda m – mantissa bo‘lib, butun qismi 1_2 ga teng. P – o‘nlik sanoq sistemasida yozilgan tartib;

3) tartibga surilish miqdori qo‘shiladi va hosil qilingan son ikkilik sanoq sistemasiga o‘tkaziladi;

4) ishorasini e’tiborga olib, sonning kompyuter xotirasidagi ko‘rinishi yoziladi.

Misol. -312,3125 sonini kompyuter xotirasidagi ko‘rinishini yozing.

Yechish. Son modulining ikkilik sistemadagi yozuvini normal ko’rinishga keltiriladi: $100111000{,}0101 = = 1{,}001110000101 \times 2^8$. Endi uning tartibi suriladi: 8 + 1023 = 1031. $1031_{10} = 10000001112$. Son manfiy bo‘lgani uchun 63 - razryadga 1 yoziladi. Natijada quyidagi son hosil qilinadi:

1	10000000111	00111000010100000000000000000...0000
63	62..52	51...0

Ayrim hollarda sonlarni ikkilik-o‘nlik sistemada ham ifodalash mumkin. Bunda to‘rt razryadli quyidagi jadvaldan foydalaniladi:

O‘nli sistemada	Ikkilik-o‘nlik sistemada	O‘nlik sistemada	Ikkilik-o‘nlik sistemada
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Bu usulda o‘nlik sanoq sistemasida berilgan sonning raqamlari to‘g‘ridan-to‘g‘ri jadvalning ikkilik-o‘nlik ustunidagi mos bitlar bilan almashtiriladi. Masalan,

$$123{,}45_{10}=0001\ 0010\ 0011,\ 0100\ 0101_{2-10}\ .$$

Ishonchli raqamlari 18 ta bo‘lgan va xotiraning 10 baytli qismida saqlanishi rejalashtirilgan butun sonlar aynan shu usulda ifodalanadi.

6.3. Harfiy ma’lumotlarni kodlash

Harfiy ma’lumotlarni kodlash uchun turli jadvallardan foydalaniladi. Biz bu holatni xalqaro ASCII jadvali (5-ilova) misolida bayon qilamiz. ASCII (American Standard Code for Information Interchange – Amerika ma’lumot almashish uchun standart kodlari) kompyuterlarda eng ko‘p qo‘llanadigan kodlash tizimlaridan biri hisoblanadi.

Bu jadvalning barcha belgilari 0 dan 255 gacha bo‘lgan sonlar bilan nomerlangan bo‘lib, ularning har biriga ikkilik sanoq sistemasidagi 8 razryadli 00000000 dan 11111111 gacha sonlar mos keladi. Aynan shu kodlar belgining ikkiliq sanoq sistemasidagi kodini (yoki o‘rnini) ifodalaydi.

ASCII jadvalining dastlabki qismi standart bo‘lib, boshqaruvchi (0 dan 32 gacha) va tinish belgilari (33 dan 47 gacha), raqamlar (48 dan 57 gacha), katta va kichik lotin harflari (65 dan 90 va 97 dan 122 gacha) kabi belgilardan tashkil topgan. Jadvalning qolgan qismi milliy alfavit (masalan, kirill alfaviti) va psevdografika elementlarini o‘z ichiga oladi.

ASCII jadvali yordamida hammasi bo‘lib 256 ta turli belgini kodlash mumkin. Bu jadvaldagi ixtiyoriy belgini 8 bit, ya’ni 1 bayt yordamida kodlash mumkin. Masalan, “Mikroprotsessor” so‘zi kompyuter xotirasida 15 bayt joyni egallaydi.

Belgilarni kodlashda ularning ASCII jadvalida turgan o‘rnidan foydalaniladi. Bu kodni ikkilik yoki o‘nlik sanoq sistemasida ifodalash mumkin. Masalan, “Hello!” so‘zi kompyuter xotirasida quyidagicha ko‘rinishda saqlanadi:

Belgi	Ikkilik	O‘nlik	Belgi	Ikkilik	O‘nlik
H	01001000	72	l	01101100	108
e	01100101	101	o	01101111	111
l	01101100	108	!	00100001	33

6.4. Kompyuter xotirasida grafik ma'elementlarni ifodalash

Grafika elementlarini ko‘rsatish uchun uchun dastlab ekran vertikal va gorizontal nuqtalar (piksellar) matritsasiga aylantiriladi. Bunda piksellar soni qancha ko‘p bo‘lsa, tasvir shunchalik aniq ko‘rinadi. So‘ngra tasvirning ifodalovchi mos piksellarga rang beriladi. Ma’lumki, ixtiyoriy rangni turli yorqinlikdagi qizil, yashil va ko‘k ranglar yordamida hosil qilish mumkin. Demak, pikselga rang berishda bu ranglarning kombinatsiyalaridan foydalaniladi.

Oq-qora tasvirlarni ifodalash uchun oq va qora ranglar yetarli. Ular xotirada bir bit yordamida ifodalanadi: 1 – oq, 0 – qora.

Rangli monitorlarda bitta piksel turli ranglarni qabul qilishi mumkin. 4 ta rangni ifodalash uchun ikki bit ma’lumot zarur bo‘ladi. Bunda quyidagi kodlash usulidan foydalanish mumkin: 00 – qora, 10 – yashil, 01 – qizil, 11 – jigarrang.

RGB-monitorlarda barcha ranglarni bazaviy ranglar (qizil, yashil va ko‘k) yordamida hosil qilinadi va ularni kodlash uchun quyidagi kombinatsiyalardan foydalaniladi:

R	G	B	Rang	R	G	B	Rang
0	0	0	qora	1	0	0	qizil
0	0	1	ko‘k	1	0	1	jigarrang
0	1	0	yashil	1	1	0	sariq
0	1	1	havorang	1	1	1	oq

Agar bazaviy ranglarning yorqinligini boshqarish imkoni mavjud bo‘lsa, bu jadvaldagi ranglar sonini orttirish mumkin. Ranglarning soni va ularni kompyuter xotirasida ifodalash uchun zarur bo‘lgan bitlar miqdori $2^N = K$ formula bilan aniqlanadi.

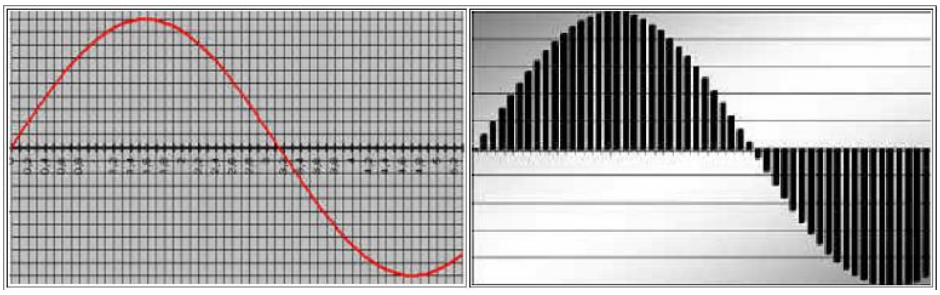
6.5. Kompyuterda tovushlarni kodlash

Fizika kursidan ma’lumki, havo zarralarining ma’lum bir amplitudada tebranishi tovushlarni hosil qiladi.

Tovushlar analogli ko‘rinishda uzluksiz o‘zgaruvchi amplituda va chastotali to‘lqinlar tarzida ifodalanadi. Bu tovushlarni bir xil vaqt oraliqlarida o‘lchab, ma’lum bir diapazondagi raqamlar ketma-ketligi tarzida ifodalanadi. Bunday metodni impulsli-kodli modullashtirish PCM (Pulse Code Modulation) deb ataladi.

Uzluksiz tovush to‘lqinlarini turli amplitudadagi ovoz impuls-lariga aylantirish kompyuterning tovush platasidagi analogli-raqamli almashtirgich qurilmasi orqali amalga oshiriladi. Zamonaviy kompyu-terlar 16 bitli ovoz kartalariga ega bo‘lib, turli balandlikdagi 65536 tovushni qayd etishga mo‘ljallangan. Bu esa tovushlarni 16 bit yordamida ikkilik sanoq sistemasida ifodalash imkonini beradi.

Tovushlar sifati vaqt birligida signallarni qayd etish chastotasiga bog‘liq bo‘ladi. Bu miqdor 8 dan 48 kGs gacha bo‘lishi mumkin.



Misol. “Kodlash chuqurligi” 16 bit, diskretlash chastotasi 48 kGs bo‘lgan va 1 minut davomida eshitiladigan yuqori sifatli stereoaudio-fayl hajmini aniqlang.

Yechish: 1 sekundli tovush faylining hajmi:

$$16 \text{ bit} \times 48\,000 \times 2 = 1\,536\,000 \text{ bit} = 187{,}5 \text{ Kbayt}.$$

Demak, 1 minutli tovushli ma’lumotning hajmi

$$187{,}5 \text{ Kbayt/s} \times 60 \text{ s} = 11 \text{ Mbayt}$$

ga teng bo‘ladi.

Takrorlash uchun savol va topshiriqlar

1. 1 bit ma’lumot nima?
2. Ma’lumotlarning qanday o‘lchov birliklarini bilasiz?
3. Sonli ma’lumotlar qanday usulda kompyuter xotirasiga yoziladi?
4. Matnlar kompyuter xotirasida qanday kodlanadi?
5. Grafik ma’lumotlarni kodlash qanday usulda amalga oshiriladi?
6. Tovushli ma’lumotlarni kodlash asosini izohlang.
7. Quyidagi ma’lumotlar kompyuter xotirasida qanday ifodalanadi:

a) 123 *b)* 23.45 *c)* −12.345

d) Stop *e)* Computer *f)* Paskal tili ?

g) “Kodlash chuqurligi” 1) 16 bit va 8 kGs ; 2) 16 bit va 24 kGs bo‘lgan 1 minutli monoaudiofayl hajmini aniqlang;

h) yuqori sifatli 10 sekundli audiofayl hajmi 940 Kbayt bo‘lsa, tovush sifatini aniqlang;

i) 16 bit yordamida kodlangan, diskretlash chastotasi 32 kGs bo‘lgan monoaudiofaylning hajmi 700 Kbayt bo‘lsa, bu ma’lumotni qancha vaqt mobaynida eshitish mumkin?

II BOB. DASTURLASH ASOSLARI

1-§. DASTUR ISHLAB CHIQISH ASOSLARI

1.1. Algoritm va unga qo‘yiladigan talablar

Inson butun hayoti davomida algoritmlar ichida yashaydi, lekin buni odatda sezmaydi. U dunyoga kelishidan tortib, to dunyodan ketishigacha bo‘lgan faoliyati davomida o‘z oldiga doim qandaydir masalalar qo‘yadi va ularni ychish yo‘l-yo‘riqlarini qidiradi. Natijada ma’lum bir qonun-qoidalarni o‘ylab topadi, belgilangan tartibda ularni bajarib, ko‘zlagan natijaga erishadi. Agar shu qonun-qoidalarni ixtiyoriy odam ko‘rsatilgan tartibda bajarishga muvaffaq bo‘lsa, u ham ana shu natijalarga erishishi, tartibni buzganda esa olgan natijalari uni qanoatlantirmasligi mumkin. Qo‘yilgan masalani yechish uchun o‘ylab topilgan qonun-qoida yoki amallar ketma-ketligi shu masalaning algoritmini tashkil qiladi.

Ko‘zlangan maqsadga erishish yo‘lida belgilangan amallar ketma-ketligini bajarayotgan inson yoki texnik vositani ijrochi deb ataymiz.

Ta’rif. *Algoritm* deb, qo‘yilgan masalani to‘la hal uchun ijrochining bajarishi lozim bo‘lgan amallar ketma-ketligining qat’iy tartibiga aytiladi.

1-misol. Ko‘chani xavfsiz kesib o‘tish qoidasi.

1. Yo‘lning chetiga kelib to‘xtang.
2. Yo‘lning chap tomoniga qarang.
3. Agar chap tomonda transport vositalari yaqin kelib qolgan bo‘lsa, o‘tib ketguncha kuting.
4. Chap tomoningizda transport vositalari qolmagan bo‘lsa, yo‘lning o‘rtasiga o‘tib to‘xtang.

5. Yo‘lning o‘ng tomoniga qarang.

6. Agar o‘ng tomonda transport vositalari yaqin kelib qolgan bo‘lsa, o‘tib ketguncha kuting.

7. O‘ng tomoningizda transport vositalari qolmagan bo‘lsa, yo‘lning qolgan qismini kesib o‘ting.

Ushbu misolda yo‘lni kesib o‘tayotgan yo‘lovchi ijrochi hisoblanadi.

Shuningdek, ixtiyoriy dorilarni ishlab chiqish yo‘llari, ovqatlarni tayyorlash usullari, shifokor belgilagan dorilarni iste’mol qilish, bankomatdan pul olish qonun-qoidalari kabi amallarni algoritm sifatida qabul qilish mumkin. Algoritmlarga turli fan sohalaridagi masalalarni yechish yo‘llari ham kiradi.

2-misol: $y = x^{20}$ ni 5 ta amal yordamida hisoblang.

1. $a1 := x * x$ x^2 hisoblandi.
2. $a2 := a1 * a1$ x^4 hisoblandi.
3. $a3 := a2 * a2$ x^8 hisoblandi.
4. $a4 := a3 * a3$ x^{16} hisoblandi.
5. $y := a4 * a2$ x^{20} hisoblandi.

Algoritmlarni qurishda quyidagi shartlarga rioya etish zarur:

1. Boshlanishi va tugashi ko‘rsatiladi.

2. Har qanday amal buyruq tarzida ifodalanadi.

3. Har bir amal ijrochiga tushunarli bo‘lgan ko‘rinishda ifodalanadi.

4. Har bir amalda qatnashayotgan o‘zgaruvchilarning qiymatlari oldindan aniqlangan bo‘ladi.

5. Har qanday amal natijasining bir qiymatli bo‘lishi ta’minlanadi.

6. Bajariladigan amallar soni cheklangan bo‘ladi.

7. Yakuniy natijalar ajratib ko‘rsatiladi va ekranga yoki qog‘ozga bosib chiqarish ta’minlanadi.

8. Qo‘yilgan masalani to‘la hal qilish uchun berilishi mumkin bo‘lgan barcha ma’lumotlar hisobga olinadi.

9. Algoritm ommaviy, ya’ni bitta sinfga taalluqli bo‘lgan ko‘plab masalalarni yechishga mo‘ljallanadi.

Yuqoridagi shartlarning birortasi buzilsa, qo‘yilgan masalani yechish uchun qurilgan algoritm to‘laqonli bo‘la olmaydi, ya’ni masalaning har qanday holatdagi to‘g‘ri yechimini bera olmaydi. Masalan, agar biron-bir amalni bajarishda qatnashayotgan bitta o‘zgaruvchining qiymati oldindan aniqlanmagan (4-shart) bo‘lsa, u holda ijrochi amalni ana shu o‘zgaruvchining o‘rniga nol qo‘yib bajaradi. Bu esa har doim ham to‘g‘ri natija beravermaydi. Faraz qilaylik, K o‘zgaruvchining qiymati oldindan aniqlanmagan bo‘lsin. U holda $D=(A+B)/K$ ifoda qiymatini hisoblashning iloji yo‘q, chunki K ning o‘rniga nol qiymati qo‘yiladi va oqibatda nolga bo‘linish holati ro‘y beradi. Nolga bo‘lish esa mumkin emas.

Endi 6-talabni buzib ko‘raylik:

1. Hisoblansin $i:=1$.
2. Hisoblansin $i:=i+1$.
3. 2-qadamga o‘tilsin.

Bunday usulda qurilgan algoritm “cheksiz algoritm” bo‘lib qoladi, yani uni “ijrochi” hech qachon tugata olmaydi.

1.2. Algoritmlarni ifodalash usullari

Algoritmlarni uch xil usulda qurish mumkin.

A) Algoritmni so‘zlar orqali qurish. Bunda algoritmning har bir buyruq-amali ijrochiga tushunarli bo‘lgan so‘zlar orqali ifodalanadi.

1-misol: AB kesmani teng ikkiga bo‘lish algoritmi.

1. Boshlansin.
2. Sirkulning bir uchi A nuqtaga qo‘yilsin.
3. Radiusi AB bo‘lgan aylana chizilsin.
4. Sirkulning uchi B nuqtaga qo‘yilsin.
5. Radiusi BA bo‘lgan aylana chizilsin.
6. Aylanalar kesishish nuqtalari va CD kesma birlashtirilsin.
7. CD va AB kesmalarning kesishish nuqtasi M belgilansin.
8. M nuqtani izlangan nuqta deb hisoblansin.
9. Ish tugatilsin.

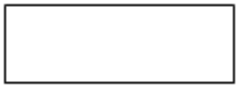
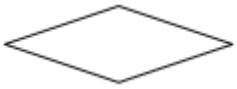
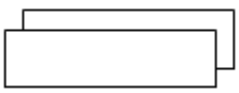
B) matematik formulalar usuli. Bu usulda algoritmning har bir amali matematik formulalar yordamida ifodalanadi. Algoritm amallarini yozishda oddiy matematik yozuvlardan foydalanish mumkin.

2-misol: $ax^2+bx+c=0$ kvadrat tenglama yechimini toping.

1. Boshlansin.
2. Aniqlansin a, b, c.
3. Hisoblansin $D := b^2+4ac$.
4. Agar $D < 0$ bo‘lsa chiqarilsin “Yechimi yo‘q” ; 7 ga o‘tilsin.
5. Hisoblansin $x_1 := \frac{-b+\sqrt{D}}{2a}$; $x_2 := \frac{-b-\sqrt{D}}{2a}$.
6. Chiqarilsin x_1, x_2.
7. Ish tugatilsin.

C) blok-sxemalar usuli. Bu usulda algoritmning har bir buyrug‘i maxsus geometrik shakllar yordamida ifodalanadi.

Blok-sxemalarni qurishda foydalanish mumkin bo‘lgan geometrik shakllar ro‘yxati quyidagi jadvalda keltirilgan.

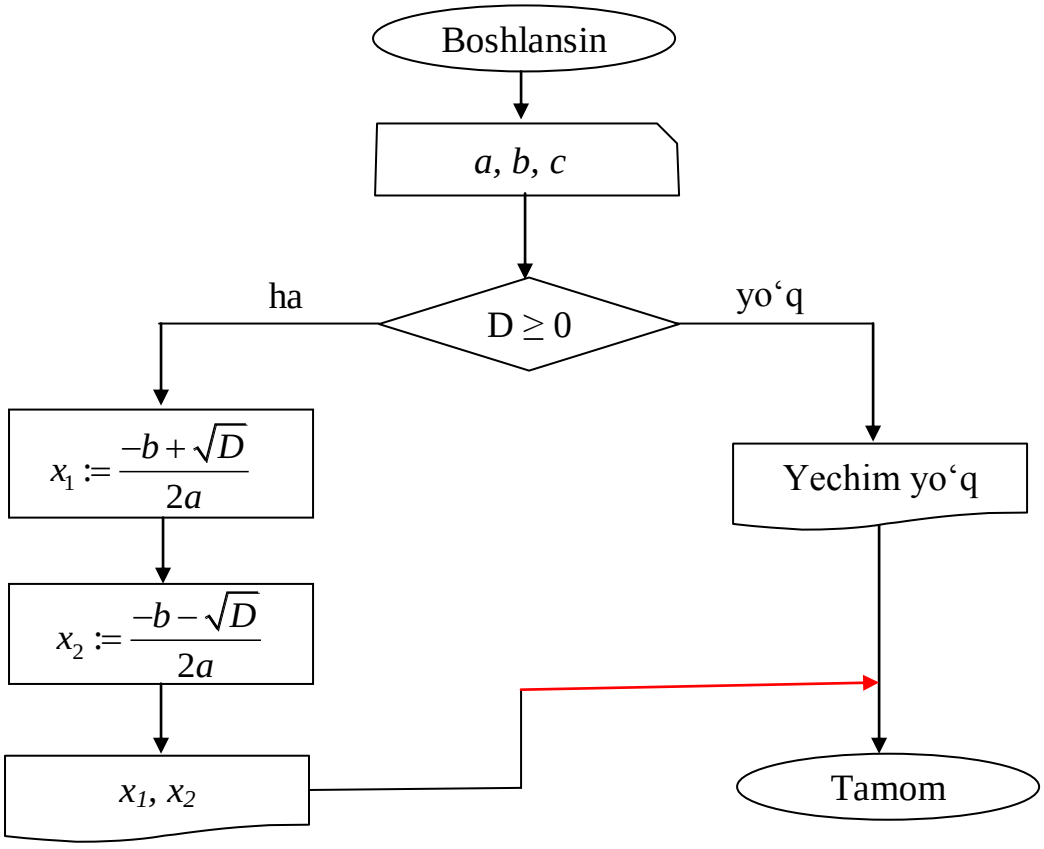
№	Shakl	Ma’nosi
1		Algoritmning boshlanishi va oxirida qo‘yiladi
2		O‘zgaruvchilarga qiymat berish
3		Ma’lumotlarni kiritish
4		Mantiqiy ifodlarning qiymatini hisoblash
5		Amallarni bajarish yo‘nalishi
6		Yordamchi algoritmlarga murojaat
7		Yakuniy natijalarni ekranga yoki qog‘ozga chiqarish

Ko‘rinishi $ax^2+bx+c=0$ bo‘lgan kvadrat tenglama uchun qurilgan blok-sxema usulidagi algoritm 1- rasmda berilgan.

1.3. Algoritmik yoki dasturlash tillari haqida

Yuqorida uch xil usulda ifodalangan algoritmlarni ko‘rdik. Ularni bilim darajasi o‘rtacha bo‘lgan ixtiyoriy ijrochi-o‘quvchi bajara oladi, chunki bu algoritmlarda uning uchun tushunarli bo‘lgan so‘z va formulalardan foydalanilgan. Informatika fani odatda algoritm ijrochisi deganda kompyuterni nazarda tutadi. Ammo u yuqorida keltirilgan usullardan birida berilgan algoritmlarni tushunmaydi. Buning oldini olish uchun algoritmlarni kompyuterga tushunarli ko‘rinishda ifodalash lozim bo‘ladi. Demak, bu yerda algoritmik til degan qo‘shimcha vositaga ehtiyoj yuzaga keladi.

Ko‘pincha algoritm va algoritmik til tushunchalari chalkashtiriladi yoki bir xil narsa deb qaraladi. Bu noto‘g‘ri.



1-rasm. Kvadrat tenglama uchun blok-sxema.

Ta’rif. Algoritmik til deb algoritmlarni ijrochiga tushunarli va bir xil ko‘rinishda ifodalash uchun zarur bo‘lgan belgilar va qonun-qoidalar majmuasiga aytiladi.

Algoritmik tillar ko‘pincha dasturlash tillari deb ham yuritiladi. Hozirgi vaqtda zamonaviy kompyuterlar uchun ko‘plab dasturlash tillari ishlab chiqilgan bo‘lib, hammasining o‘ziga xos afzalliklari, imkoniyat hamda qonun-qoidalari mavjud. BASIC, Turbo Pascal, Fortran, C, C++ dasturlash tillari ana shular jumlasidandir.

Dasturlash tillari imkoniyatlarining turlichaligi bilan bir-biridan farq qiladi. Masalan, BASIC algoritmik tili o‘rganish uchun sodda va qulay bo‘lib, unchalik murakkab bo‘lmagan muhandislik masalalari uchun mo‘ljallangan. C++ tili esa zamonaviyligi, dastur yozish jarayonida yo‘l qo‘yilishi mumkin bo‘lgan xatoliklarning oldini olish, yangi tipdagi funksiya va ma’lumotlarni hosil qilish, rekursiv funk-siyalar bilan ishlash, grafik imkoniyatlarining kengligi va boshqa ko‘plab xususiyatlari bilan boshqa tillardan farq qiladi. Bundan tashqari, bu til hozirgi kunda muxlislari tobora ko‘payib borayotgan zamonaviy Vizual C, C++ Builder va C# kabi dasturlash muhitlarini o‘rganish uchun asosiy poydevor hisoblanadi. Shuni ham alohida ta’kidlash joizki, bugungi kunda eng yangi kompyuter va kompyuter tizimlari uchun dasturiy mahsulotlarning kattagina qismi C++ das-turlash tili yordamida ishlab chiqilmoqda.

1.4. C++ dasturlash tili haqida

C++ dasturlash tili C ning rivojlangan varianti sanaladi. O‘z nav-batida, C dasturlash tili BCPL va B tillari asosida ishlab chiqilgan. BCPL tili 1967-yili Martin Richard tomonidan yaratilgan.

C dasturlash tili dastlab Denis Ritchi tomonidan rivojlantirildi va ilk marta 1972-yil DEC PDP-11 kompyuterida sinaldi. Dastlab C dasturi Unix operatsion tizimida qayta ishlovchi til sifatida mashhur bo‘ldi. Bugungi kunda ko‘pgina operatsion tizimlar va ularga mo‘ljal-langan dasturiy ta’minot C va C++ dasturlarida ishlab chiqilmoqda.

C dasturlash tili 70- yillarning oxirlariga kelib jadal rivojlandi. Hozirgi kunda uning “an’anaviy”, “klassik” yoki “Kernigan va Ritchi” turlari mavjud.

C tilining turli kompyuterlarda keng qo‘llanilishi bu tilning o‘zaro o‘xshash, lekin bir-biriga bog‘liq bo‘lmagan variantlarining (versiyalarining) ishlab chiqilishiga olib keldi. Bu esa C tilidagi yangiliklarni o‘zlashtirish va dastur ishlab chiqish sohasida dasturchilar uchun turli noqulayliklarning yuzaga kelishiga sabab bo‘ldi.

80-yillarning boshlarida dasturchilarga Byern Stroustrop tomonidan C dasturlash tilining kengaytirilgan C++ varianti taklif qilindi. 1983-yili Amerika Milliy Standart komitetida hisoblash texnikasi va axborotlarni qayta ishlash sohasida X3J11 texnik komiteti tuzildi. 1989-yili esa standarti tasdiqlangan C dasturlash tili 1990-yilda dunyo bo‘yicha standartlashtirildi.

C++ dasturlash tili C tilini “tartibga keltiruvchi” xossasiga ega hamda obyektga asoslangan dasturlashga imkon beradi. Bunday imkoniyatning yuzaga kelishi dasturiy ta’minot ishlab chiqish olamida tub o‘zgarishlarning ro‘y berishiga olib keldi.

Ushbu dasturlash texnologiyasi asosida obyektlar yotadi va ular o‘zida real hayot elementlarini mujassamlashtiradi. Aynan shu sababli obyektga asoslangan dastur matnlarini ishlab chiqish, tushunish va tahrirlash amallari osonlashdi.

Hozirgi kunga kelib, bir qator obyektga asoslangan dasturlash tillari ishlab chiqilgan va amaliyotda keng qo‘llanilmoqda. Ular orasida Visual Basic, Deplhi, Java, C++, C++ Builder kabi dasturlash tillarini alohida ta’kidlab o‘tish mumkin. Bu tillarning har biri yechilayotgan masala xarakteriga ko‘ra boshqasidan afzalloq bo‘lishi mumkin, sodda qilib aytganda, ba’zi masalalar uchun bu tillarning biri qulay bo‘lsa, boshqa masala uchun ikkinchisi qulay hisoblanadi. Shuning uchun obyektga asoslangan dasturlash tillarining birortasiga mutloq ustunlikni berish mantiqan to‘g‘ri bo‘lmaydi. Ular ikki raqobatchi kompaniya Microsoft (Visual Basic, Java tillarini ishlab

chiqqan) hamda Borland (C++, Delphi tillarini ishlab chiqqan) o‘rtasidagi kurash mobaynida shakllandi, rivojlandi va imkoniyatlari tobora kengayib bormoqda. Bu raqobat hozircha to‘xtaganicha yo‘q. Shuning uchun “Qaysi dasturlash tili yaxshi?” degan savolga javob tariqasida “Siz qanday masalani yechmoqchisiz?” deyish o‘rinli bo‘lar edi.

Takrorlash uchun savol va topshiriqlar

1. Algoritm nima?
2. Algoritmning asosiy xossalarini ayting.
3. Algoritmik til nima?
4. Quyidagi masalalar uchun blok-sxema va matematik formula ko‘rinishidagi algoritmlar ishlab chiqing:

a) Bir tomoni va unga yopishgan ikki burchagi bo‘yicha uchburchak yuzini toping.

b) Ikki tomoni A va B orasidagi burchagi γ ga teng bo‘lgan uchburchakning noma’lum burchaklari va perimetri topilsin.

c) $Ax^3 + Bx = 0$ tenglamani yeching.

d) N natural son berilgan bo‘lsin. $1+2+3+...+N$ yig‘indini hisoblang.

2-§. C++ HAQIDA BOSHLANG‘ICH MA’LUMOTLAR

2.1. C++ tilining alifbosi

C++ dasturlash tilining alifbosi deb, shu tilda ma’lumotlarni ifodalash va dastur ishlab chiqish jarayonida kompilyator tomonidan qabul qilishga ruxsat berilgan belgilar yoki maxsus belgilardan iborat zanjirlar to‘plamiga aytiladi.

Bu alifbo ASCII (xalqaro belgilar va ularning kodlari) jadvalining hamma belgilarini, ya’ni quyidagilarni o‘z ichiga oladi:

1) lotin alifbosining katta va kichik harflari;

2) 0 dan 9 gacha arab raqamlari;

3) tagiga chizish belgisi (_);

4) bo‘sh joy belgisi;

5) boshqaruvchi belgilar: ASCII jadvalidagi (1-ilovaga qarang) kodlari 0 dan 31 gacha bo‘lgan belgilar. Bu belgilar satr va konstantalarni ifodalashda qo‘llanishi mumkin;

6) turli ko‘rsatmalarni yozish uchun ishlatiladigan maxsus belgilar:

+ – * / = { } () [] . , @ # ' : ; $ % ^ ! ? & \ < >

7) asosiy bo‘lmagan belgilar (ASCII ni kengaytiruvchi, ya’ni kodi 128 dan 255 gacha bo‘lgan belgilar; rus alifbosining katta va kichik harflari, psevdografika elementlari shu sinfga kiradi. Bu belgilar turli konstantalarni hosil qilish, matnlarni yozish, izohlarni tashkil qilishda qo‘llanishi mumkin);

8) murakkab belgilar: <= >= := .. * *;

9) xizmatchi so‘zlar. Ular C++ dasturlash tilida ma’lum bir ma’no yoki ko‘rsatmani anglatuvchi maxsus belgilar zanjiridan iborat bo‘lib, bu zanjirni o‘zgartirish yoki qisqartirib qo‘llash mumkin emas. Masalan: *main, include, iostream, int* va hokazo.

Eslatma. Agar dastur tarkibida yuqorida sanab o‘tilgan belgilardan boshqa belgi yoki xizmatchi bo‘lmagan so‘zlar uchrab qolsa, bu haqidagi maxsus axborot kompyuter ekraniga chiqariladi.

2.2. Ma’lumotlarning tiplari

C++ tilidagi dastur masala xarakteridan kelib chiqqan holda kutilgan natijaga ega bo‘lish uchun berilgan ma’lumotlar ustida ma’lum bir amallarni bajarishni ta’minlaydi. Bu ma’lumotlar asosan bazaviy hisoblangan 7 ta tipdan (1-jadval) biriga mansub bo‘lishi mumkin.

1-jadval

Bazaviy ma’lumot tiplari

bool	mantiqiy tip
char	belgili (harfiy) tip
wchar_t	ikki baytli belgili (harfiy) tip

double	qo‘sh aniqlikdagi haqiqiy sonli tip
float	haqiqiy sonli tip
int	butun sonli tip
void	qiymatga ega bo‘lmaydi

C++ tilida bu tiplarni *signed* (ishorali), *unsigned* (ishorasiz), *short* (qisqartirilgan) va *long* (kengaytirilgan) kabi modifikatorlar yordamida o‘zgartirish imkoniyati ham ko‘zda tutilgan. Bu modifikatorlarning barchasi bilan *int* tipidagi ma’lumotlar, *signed* va *unsigned* orqali *char*, *long* bilan esa *double* tipidagi ma’lumotlarni o‘zgartirish mumkin.

Hosil qilingan barcha tipdagi ma’lumotlar turli kompilyatorlar uchun kompyuter xotirasidan turli hajmdagi joyni band qiladi. 2-jadvalda ma’lumotlar tiplariga ko‘ra egallashi mumkin bo‘lgan minimal joy standartlari 32-razryadli muhit uchun keltirilgan.

2- jadval

Ma’lumot tiplari uchun minimal diapazon

tip	bit	qiymatlar diapazoni
bool	1	false yoki true
chart	8	-128 dan 127 gacha
wchar_t	16	0 dan 65635 gacha
double	64	2.2E−308 dan 1.8E+308 gacha
float	32	1.8E−38 dan 1.8E+38 gacha
int	32	−2147483648 dan 2147483647 gacha
unsigned char	8	0 dan 255 gacha
signed char	8	−128 dan 127 gacha
unsigned int	32	0 dan 4294967295 gacha
signed int	32	−2147483648 dan 2147483647 gacha
short int	16	-32768 dan 32767 gacha
unsigned short int	16	0 dan 65535 gacha

signed short int	16	-32768 dan 32767 gacha
long int	32	−2147483648 dan 2147483647 gacha
unsigned long int	32	0 dan 4294967295 gacha
signed long int	32	−2147483648 dan 2147483647 gacha

Eslatma: *void* tipidan qiymatga ega bo‘lmaydigan funksiyalarni aniqlashda foydalaniladi.

Aslini olganda, 2 va 2.0 sonlari bir xil miqdorni anglatadi. Lekin C++ tili kompilyatori ularni bir-biridan farqlaydi, ya’ni 2 sonini butun, 2.0 ni esa haqiqiy son deb qabul qiladi.

Butun sonlar bilan kompilyator bo‘lish amalidan tashqari barcha amallarni bajara oladi. Ammo haqiqiy sonlar bilan ishlaganda, yax-litlash hisobiga taqribiylikka yo‘l qo‘yishi mumkin. Shuning uchun ma’lumotlar tipini aniqlashga jiddiy yondoshish zarur.

Char tipidagi ma’lumotlar odatda bitta belgi yoki harfdan iborat bo‘ladi va ularni apostrof (“ ’ ” – yakka qo‘shtirnoq) belgisi bilan ko‘rsatiladi: ‘a’, ‘d’. ***Wchar_t*** tipidagi ma’lumotlar esa L harfi bilan birgalikda yoziladi: L’a’, L’d’ va h.k.

2.3. C++ tilida o‘zgaruvchi va o‘zgarmaslar

Ma’lumki, dasturlar turli sonli va boshqa tipdagi ma’lumotlarni qayta ishlash uchun yoziladi. Bitta dastur tarkibida uchraydigan ma’lumotlarni bir-biridan farqlash uchun nomlash lozim bo‘ladi va bunday nomlar ***identifikator*** deb ataladi.

Identifikator muayyan bir vaqtda ifodalab turgan son yoki bosh-qa turdagi ma’lumot uning qiymati hisoblanadi. Dasturda qatnashgan har bir identifikator uchun kompyuter xotirasidan ma’lum bir joy ajratiladi hamda bu joyga uning qiymati yozib qo‘yiladi va saqlanadi.

Identifikatorlar doim lotin harflari bilan boshlanadi. Ularni yozish uchun zarur bo‘lgan keyingi belgilar esa lotin harflari, raqamlar va “_” (tagiga chizish) belgisidan iborat bo‘lishi mumkin:

X, xl, s4, absl2d, fam, kitob_soni.

Dastlabki to‘rtta identifikator sintaktik jihatdan to‘g‘ri yozilgan,

ammo identifikatorlarni bunday yozish katta hajmli dasturlarni ishlab chiqishda ma’lum bir qiyinchiliklarni tug‘dirishi mumkin, chunki ular o‘zlari ko‘rsatayotgan ma’lumotlarni to‘la va tushunarli qilib ifodalay olmaydi. Natijada bunday identifikatorlarni boshqasi bilan almashtirib yuborish ehtimolligi ortadi hamda ana shu dasturni o‘qish va tushunish qiyinlashib ketadi. Shuning uchun identifikatorlarni keyingi ikkitasi kabi belgilash maqsadga muvofiq hisoblanadi. Chunki ular o‘zlari ifodalab turgan ma’lumotlarni ma’lum bir darajada izohlaydi va shu bilan bog‘liq anglashilmovchiliklarga barham berishda muhim ahamiyat kasb etadi.

Identifikator tanlaganda ma’lumotlarning shakli va mazmunini hisobga olish ham ana shunday omillardan biri hisoblanadi. Masalan, uchburchak haqidagi masalada ehtiyojga qarab

a_tomon, b_tomon, c_tomon, yarim_perimetr, yuza

kabi identifikatorlar maqsadga muvofiq hisoblanadi.

Identifikatorlarni yozishda C++ tili kompilyatori katta va kichik harflarni bir-biridan farqlaydi, ya’ni *yuza, Yuza, YuZA, YuZa* kabi identifikatorlarni turli xil deb qabul qiladi.

Identifikator sifatida xizmatchi so‘zlar, turli tinish va munosabat belgilaridan foydalanib bo‘lmaydi. Shuning uchun ularni quyidagicha yozish noto‘g‘ri hisoblanadi:

$$4x,\ X-y,\ G=dr,\ AB,\ !gamma,\ a?b,\ begin,\ end.$$

Agar identifikator dasturning bajarilishi davomida o‘z qiymatini o‘zgartirmasa, ularni o‘zgarmaslar yoki konstantalar, aks holda o‘zgaruvchilar deb ataladi.

O‘zgarmas ma’lumotlar dastur matnida maxsus xizmatchi ***const*** so‘zi yordamida alohida ta’kidlab ko‘rsatiladi. Masalan:

$$const\ \ float\ \ gamma=1.23;.$$

Shundan keyin bunday o‘zgarmas qiymatlarni dasturning bajarilishi davomida o‘zgartirib bo‘lmaydi.

Dastur tarkibida uchraydigan hamma identifikatorlarga kompyuter xotirasidan joy ajratiladi va bu joyda ularning qiymatlari saqlanadi.

Bu identifikatorga murojaat qilinganda, u uchun ajratilgan joy, ya’ni yacheykada saqlanayotgan ma’lumot o‘qiladi va bu ma’lumot identifikator o‘rniga qo‘yiladi.

Identifikator uchun xotiradan ajratilayotgan joy hajmi uning tipiga bog‘liq bo‘ladi.

2.4. Sonlar, arifmetik amallar va ifodalarni yozish

Musbat va manfiy butun sonlarni C++ tilida ham odatdagidek yozish mumkin:

Oddiy yozuv	**C++ tilida**
1	1
-1024	-1024

Haqiqiy sonlarni yozishda, ularning butun va kasr qismini ajratib ko’rsatish uchun odatdagi vergul o‘rniga nuqta belgisidan foydalaniladi.

Oddiy yozuv	**C++ tilida**
1,0	1.0
-1,024	-1.024

Ko‘chuvchi vergulli yozuvda esa haqiqiy sonning butun va kasr qismini ajratib turuvchi vergulni ehtiyojga qarab o‘ngga yoki chapga surish mumkin. Bunda vergulni surish xonalariga ko‘ra sonning tartibi o‘zgaradi. Masalan:

$$123.0 = 12.0 \cdot 10^{1} = 0.123 \cdot 10^{3} = 12300.0 \cdot 10^{-2}.$$

Bunday shaklda yozilgan sonlarni eksponentsial ko’rinishdagi sonlar deb ham yuritiladi. C++ tilida eksponentsial sonlarni yozish uchun odatdagi “o‘nning darajasi” yozuvi o‘rniga “e” belgisidan foydalaniladi.

Oddiy yozuv	**C++ tilida**
$1,2 \cdot 10^{25}$	1.2e25

$1,23\cdot10^{-19}$	1.23e-19
$-2,8\cdot10^{-27}$	-2.8e-27

C++ tilida sonli ma’lumot va o‘zgaruvchilar ustida qo‘shish (+), ayirish (-), ko‘paytirish (∗), bo‘lish (/), qoldiqni aniqlash (%) kabi amallarni bajarish mumkin.

Arifmetik amallar odatdagi yozuvdan bitta satrga yozilishi bilan farqlanadi.

Oddiy yozuv	**C++ tilida**
$a+b$	$a+b$
b^2-4ac	$b*b-4*a*c$
$z(a+b)$	$z*(a+b)$
$a:b$	a/b
$\frac{x-y}{x+y}$	$(x-y)/(x+y)$
$\frac{x}{1+\frac{c+d}{nm}}$	$x/(1+(c+d)/(n*m))$

Arifmetik ifodalarda standart funksiyalar tez-tez uchrashini hisobga olib, ularning ayrimlarini C++ da yozilishini keltirib o‘tamiz. Bu funksiyalarning argumentlari qavslar ichida yoziladi.

Oddiy yozuv	**Ma’nosi**	**C++ tilida**
$\lvert x\rvert$	absolut qiymat	***abs(x)***
$\sqrt{x}$	kvadrat ildiz	***sqrt(x)***
e^x	eksponenta	***exp(x)***
$\ln x$	natural logarifm	***log(x)***
$\sin x$	sinus	***sin(x)***

$\cos x$	kosinus	***cos(x)***
$\mathrm{tg}\, x$	tangens	***tan(x)***
$\arccos x$	arkkosinus	***acos(x)***
$\arcsin x$	arksinus	***asin(x)***
$\mathrm{arctg}\, x$	arktangens	***atan(x)***

Eslatma. 1. Jadvaldagi trigonometrik funksiyalarning argument-lari radianlarda ifodalanishi lozim.

2. Jadvalga kirmagan funksiyalarni jadvaldagi funksiyalar orqali ifodalash shart. Masalan:

$\log_a b = \frac{\ln b}{\ln a}$ yoki $\mathrm{ctg}\, x = \frac{\cos x}{\sin x}$ kabi.

1-misol. Quyidagi ifodalarni C++ tilida yozing.

$a)\ \frac{x+y}{a_1}\cdot\frac{a_2}{x-y}+\sqrt{\sin 2\alpha}$ $b)\ \log_a(b+c)+\left|(a+bx)\beta\right|$

Yechish:

Oddiy yozuv **C++ tilida**

$a)\ \frac{x+y}{a_1}\cdot\frac{a_2}{x-y}+\sqrt{\sin 2\alpha}$ *(x+y)/a1*a2/(x-y)-sqrt(sin(2* alfa))*

b) $\log_a(b+c)+|(a+bx)\beta|$ *ln(b+c)/ln(a)+abs((a+b*x)*beta)*

2-misol. C++ tilida berilgan yozuvlardagi xatoliklar aniqlansin.

*a)(x+y)/0.0-sqr(sin(alfa)*cos(beta)*

b)abs(exp(x)-sin (sqr(a+b)/gamma)+sqrt(abs(x))

Takrorlash uchun savol va topshiriqlar

1. Identifikator nima?
2. Ma’lumotlarning qanday tiplarini bilasiz?
3. ***Float*** tipidagi ma’lumotlarning diapazoni qanday?
4. ***Char*** tipidagi ma’lumot nima?
5. ***Int*** tipini ***long*** orqali o‘zgartirib, yana qanday tiplarni hosil qilish mumkin?
6. Quyidagi yozuvlarni C++ tilida yozing:

$$a)\ u=(1+z)\cdot\frac{x+\frac{y}{z}}{a-\frac{1}{1-x^2}}+\frac{\sin^2 x}{x^2+y^2};$$

$$b)\ y=\frac{\sqrt{\left|\ln|x-ab|-x^3\right|}}{\sin\alpha+\sin\beta}+\sqrt[3]{\frac{3VH^2}{\pi r^2}}+\frac{1}{3}.$$

3-§. SODDA DASTURLAR YOZISH

3.1. Dasturning umumiy ko‘rinishi

C++ tilida dasturlarning umumiy ko‘rinishi quyidagicha:

```
# include <iostream.h>
int  main()
{ ...;
}
```

Bu yerda keltirilgan xizmatchi so‘zlar quyidagi ma’nolarni anglatadi.

# include	Dasturning majburiy elementi bo‘lib, qavslar orasida ko‘rsatilgan modullar kutubxonasini ishga tushirishga ko‘rsatma beradi
<iostream.h>	Ma’lumotlar oqimini kiritish va chiqarish jarayinonini tashkil qilish kutubxonasi
int main()	Dastur matni odatda shu xizmatchi so‘zdan keyin boshlanadi. C++ tilidagi har bir dastur ***main()*** fuksiyasini o‘zida saqlaydi. Funksiya bu bir yoki bir necha amalni bajaruvchi dastur blogi hisoblanadi. Odatda dasturda bir funksiya boshqasi orqali chaqiriladi, lekin ***main()*** funksiyasi alohida xususiyatga ega bo‘lib u dastur ishga tushirilishi bilan avtomatik tarzda chaqiriladi.
{ ***...*** ***}***	Dastur kodi to‘laligicha (yoki bir parchasi) figurali qavslar (**{…}**) orasida yoziladi. Bu qavslarni operatorlar qavsi deb ham ataladi. Operatorlar blogini bajarish tartibi ular joylashgan qavslar holati bilan aniqlanadi.

;	Dastur kodini tashkil etuvchi barcha buyruqlar bir-biridan nuqtali vergul bilan ajratiladi.

3.2. Qiymat berish buyrug‘i

O‘zgaruvchilar ko‘rsatayotgan ma’lumotlarni o‘zgartirish uchun ularga qiymat beriladi. Bu amalni qiymat berish buyrug‘i, ya’ni “=” belgisi yordamida amalga oshiriladi.

Buyruqning umumiy ko‘rinishi quyidagicha:

$$\alpha = \beta;$$

bu yerda α – qiymat olayotgan o‘zgaruvchi, β esa qiymati α ga beriladigan sonli, arifmetik, mantiqiy yoki harfiy ifoda. Bu buyruqning ma’nosi quyidagicha: β ifodaning qiymati hisoblanadi va bu qiymat α ga beriladi, ya’ni xotiradan α uchun ajratilgan yacheykaga yozib qo‘yiladi. Masalan:

$$x = 20.25;\ y = (x + 0.75) * 2;\ a1 = 'C++\ tili';$$

Bu buyruqlar bajarilganidan so‘ng, ***x*** o‘zgaruvchi 20,25 ni, ***y*** esa 42 ni, harfiy ***a1*** o‘zgaruvchi esa “C++ tili” degan qiymatni oladi. Demak, bu o‘zgaruvchilar uchun ajratilgan yacheykaga ana shu qiymatlar yozib qo‘yiladi.

Agar zarur bo‘lsa, qiymat olayotgan o‘zgaruvchi qiymat berish buyrug‘ining o‘ng tomonida ham kelishi mumkin. Bu holda o‘ng tomondagi ifodaning qiymatini hisoblash uchun uning “eski” qiymatidan foydalaniladi. Ifodaning “yangi” qiymati hisoblab topilganidan keyin, yacheykadagi “eski” qiymat o‘chiriladi va uning o‘rniga yangisi yozib qo‘yiladi. Masalan:

```
alfa=15;  alfa=alfa*2;
```

buyruqlari bajarilganidan keyin, ***alfa*** ning qiymati 30 ga teng bo‘lib qoladi.

C++ tilida postfiks ko‘rinishidagi qiymat berish buyrug‘idan ham foydalanish mumkin. Unga ko‘ra oxirgi buyruqni

```
alfa*=2;
```

ko‘rinishida ham yozish mumkin. Bu buyruqlarning har ikkisi ekvivalent hisoblanadi.

Qiymat berish buyrug‘ining o‘ng tomonidagi ifodada qatnasha-

yotgan har bir o‘zgaruvchining qiymati oldindan aniqlangan (initsializatsiya qilingan) bo‘lishi lozim. Aks holda uning o‘rniga nol soni qo‘yiladi va ifoda ana shu qiymat uchun hisoblanadi.

C++ tilida an’anaviy arifmetik amallardan tashqari inkrement va dekrement deb ataluvchi buyruqlardan ham foydalanish ko‘zda tutilgan.

$i++$ ko‘rinishida yoziladigan inkrement buyrug‘i o‘zgaruvchining (to‘g‘rirog‘i *i* ning) qiymatini 1 ga orttirishni anglatadi va u an’anaviy usulda yoziladigan $i = i + 1$ buyrug‘i bilan ekvivalent hisoblanadi. Bu buyruqni C++ tilida $++i$ tarzidagi postfiks ko‘rinishida ham yozish mumkin.

$i--$ ko‘rinishida yoziladigan dekrement buyrug‘i *i* ning qiymatini 1 ga kamaytirishni bildiradi va u an’anaviy usulda yoziladigan $i = i - 1$ buyrug‘i bilan bir xil ma’noni anglatadi. Bu buyruqni C++ tili postfiks ko‘rinishida, ya’ni $--i$ qabilida yozishga ham ruxsat beradi. Shuning uchun

$$i = i + 1;\ \ i++;\ \ ++i;$$

buyruqlari bir-biriga ekvivalent. Shuningdek,

$$i = i - 1;\ \ i--;\ \ --i:$$

buyruqlari ham bir xil ma’noni anglatadi.

3.3. Ma’lumotlarni chiqarish buyrug‘i

Ko‘pincha masalalarni yechish jarayonida masalaning shartida talab qilingan ma’lumotlarni ekranga (yoki qog‘ozga) chiqarishga to‘g‘ri keladi.

Cout operatori o‘zgaruvchilarning qiymatlari, matnlar hamda arifmetik ifodalarning qiymatlarini hisoblab displey ekraniga chiqarish uchun xizmat qiladi. Bu operator umumiy holda quyidagicha yoziladi:

cout << (ekranga chiqariladigan ma’lumotlar ro‘yxati)

Ekranga chiqarish kerak bo‘lgan matnlar qo‘shtirnoq belgisi bilan ko‘rsatiladi. Masalan:

cout << **“***C++ tili”**;*

buyrug‘i ekranga

C++ tili

yozuvini chiqaradi.

Cout operatori yordamida C++ tili qoidalari bilan yozilgan arifmetik ifodalarning qiymatlarini ham hisoblash mumkin. Masalan:

cout<< 3*4+15/3-10.5 ;

buyrug‘i bajarilganda, 3·4+15/3-10.5 ifodaning qiymati hisoblanadi va natija 6.5 ko‘rinishda ekranga chiqariladi.

Agar *cout* yordamida ekranga chiqarish talab qilingan ma’lumotlar sifatida o‘zgaruvchilar ro‘yxati berilgan bo‘lsa, u holda bu o‘zgaruvchilarning qiymatlari xotiradan qidirib topiladi va ekranga chiqariladi. Faraz qilaylik, biror dasturning bajarilishi davomida *x, y, z* o‘zgaruvchilar mos ravishda 23, 123.45, ‘S’ qiymatlarini olgan bo‘lsin. U holda

cout <<x<< y<<z;

operatorining bajarilishi natijasida ekranda

23123.45S

ko‘rinishidagi ma’lumotlar paydo bo‘ladi. Ko‘rinib turibdiki, ekranga uzatilgan ma’lumotlar yonma-yon yoziladi va bu holat natijalarni tahlil qilishda anglashilmovchilikka olib keladi. Buning oldini olish uchun chiqarilishi lozim bo‘lgan ma’lumotlar o‘rtasiga “bo‘sh joy” belgilarini kiritish lozim bo‘ladi. Bosqacha aytganda, *cout* buyrug‘ini quyidagicha yozish tavsiya etiladi:

cout <<x<< “ “<< y<<“ “<< z;

Bu buyruq ta’sirida ma’lumotlar quyidagi ko‘rinishda ekranga chiqariladi:

23 123.45 S

Eslatma: Kompilyator haqiqiy sonlarni yaxlitlashi mumkinligini e’tiborga olinsa, bu buyruqning natijasi ekranda

23 123.449997 S

ko‘rinishida ham paydo bo‘lishi mumkin.

Ekranda kursor-ko‘rsatkich mavjud bo‘lib, u ma’lumotlarning qaysi pozitsiyadan boshlab chiqarilishini ko‘rsatadi. Navbatdagi ma’-

lumot kursor turgan joydan boshlab chiqariladi.

Cout operatori oxirida turgan “*n*” ko‘rsatmasi ro‘yxatda ko‘rsatilgan ma’lumotlarni ekranga chiqarilgandan keyin, kursorni yangi satr boshiga o‘tkazadi. Masalan:

cout<<x<<” “; cout<<y<<” “; cout<<z<<” “;

buyruqlari ma’lumotlarni

23 123.45 S

ko‘rinishida ekranga chiqarsa,

cout<<x<<” \n“; cout<<y<<” \n“; cout<<z;

buyruqlari ma’lumotlarni ekranga

23
123.45
S

tarzida chiqarilishini ta’minlaydi.

3.4. Chiziqli dasturlar yozish

Chiziqli yoki sodda dastur deganda ko‘rsatilgan buyruqlar ketma-ketligi dasturda uchrash tartibiga mos ravishda bajariladigan dasturlar tushuniladi. 3.1-bandda C++ tilidagi dasturlarning umumiy ko‘rinishi keltirilgan edi:

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

Dasturga majburiy bo‘lmagan, ammo ehtiyoj paydo bo‘lganda yozish shart bo‘lgan yangi qo‘shimcha elementlarni qo‘shish mumkin. Biz bu elementlar bilan keyinchalik tanishamiz.

C++ tilidagi dastur (kod) qo‘yilgan masala algoritmidagi buyruqlarni yuqoridagi umumiy tuzilmaga muvofiq kompilyatorga “tushunarli” bo‘lgan ko‘rinishda ifodalash natijasida hosil bo‘ladi.

Demak, chiziqli dasturlarda dasturni bajarish jarayoni boshlan-

ganda, dastlab 1-buyruq, keyin 2-buyruq va h.k. bajariladi. Bunda “{“ va “}” belgilarga alohida e’tibor beriladi. Ular operatorlar qavsi hisoblanib, amallarni bajarish paytida ustunlikka ega bo‘ladi.

Dastur matnidagi har bir buyruq yoki ko‘rsatma bir-biridan “ ; ” belgisi bilan ajratiladi. Dasturning bitta satrida bitta yoki bir nechta buyruqlar kelishi mumkin. (Ammo bu holat dasturlashni endigina o‘rganishni boshlagan dasturchilarga tavsiya etilmaydi.) Bu holda ham har bir buyruq bir-biridan “ ; ” belgisi bilan ajratiladi.

1-masala. Ekranga “Salom C++” matnini chiqaruvchi dastur yozing.

Yechish g‘oyasi. Ushbu masalaning yechimi juda ham sodda bo‘lib, ortiqcha izohga hojat yo‘q. Keltirilgan umumiy tuzilmaga ekranga “Salom C++” matnini chiqarishni ta’minlovchi bitta *cout* buyrug‘ini qo‘shish yetarli. Shuning uchun qo‘yilgan masalaning kodi quyidagicha yoziladi:

```
# include <iostream.h>
int main()
{
cout<<”Salom C++”;
return 0;
}
```

Umuman aytganda, C++ tilida dastur kodlarini kompyuter yordamida hal qilish kamida uchta bosqichdan iborat bo‘lishi mumkin:

1) TC dasturlash muhiti ishga tushirilganidan so‘ng dastur kodi kompyuter xotirasiga kiritiladi;

2) bu kod kompilatsiya qilinadi. Kompilatsiya jarayonida dastlab kodda mavjud bo‘lgan sintaktik xatolar bartaraf etiladi, so‘ngra kod mashina tiliga o‘giriladi va *.cpp* kengaytmali faylda saqlab qo‘yiladi. Kompyuter bu amallarni bajarishi uchun dasturchi TC ning bosh menyusidagi *Compile* punktidagi *Compile* buyrug‘i yoki *Alt+F9* tugmalarini bosishi lozim;

3) kompilatsiya jarayoni muvaffaqiyatli tugallanganidan so‘ng, dastur kodini ishga tushirish mumkin. Buning uchun TC bosh menyusi

Run punktidagi *Run* buyrug‘i yoki *Ctrl+F9* tugmalarini bosish lozim.

Dastur natijasini ko‘rish uchun *Alt+F5* tugmalari bosiladi.

2-masala. Uzunligi 25 sm bo‘lgan aylana bilan chegaralangan doira yuzini toping.

Yechish g‘oyasi. Ma’lumki, doira yuzi $S = \pi R^2$ formula bilan hisoblanadi. Bu formulani qo‘llashdan oldin R ning qiymatini topish talab qilinadi. Uni esa aylana uzunligining 25 sm ekanligidan topish mumkin, ya’ni $R = 25/(2\pi)$.

Bu ma’lumotlarni hisobga olib, qo‘yilgan masala algoritmi va dasturini quyidagicha yozish mumkin:

Algoritmi

1. Boshlansin.
2. Hisoblansin $L := 25$.
3. Hisoblansin $R = L/(2\pi)$.
4. Hisoblansin $S = \pi R^2$.
5. Chiqarilsin *S*.
6. Ish tugatilsin.

Dastur kodi

```
# include <iostream.h>
int main()
{  float L, R, pi, S;
    pi=3.14; L=25;
    R=L/(2*pi);
    S=pi*R*R;
    cout<<”S=”<<S;
  return 0;  }
```

Ushbu kodning 3-satrida *L, R, pi, S* o‘zgaruvchilarning tiplari ko‘rsatilgan. Bu amalni o‘zgaruvchilarga birinchi marta qiymat berish jarayonida ham bajarish mumkin. Bunday usulda kod ishlab chiqish uchun 3- va 4-satrlarni

float pi=3.14; float L=25

qabilida o‘zgartirish mumkin.

Ushbu kod 1-masaladagi kabi kompyuterda bajariladi va u ekranga

S = 49.761147

yozuvini uzatadi.

3.5. Ma’lumotlarni klaviaturadan kiritish

Ko‘pincha sharti 2-masaladagi kabi aniq emas, balki umumiy

bo‘lgan holat uchun masalalarni hal qilishga to‘g‘ri kelib qoladi.

3-masala. Uzunligi *L* bo‘lgan aylana bilan chegaralangan doira yuzini toping.

Bunday hollarda masala shartida berilgan *L* ning qiymatini klaviatura orqali kiritishga to‘g‘ri keladi. Bu muammoni C++ tilida umumiy ko‘rinishi

cin >> o‘zgaruvchilar ro‘yxati

bo‘lgan operator yordamida hal qilish mumkin.

Ro‘yxatdagi o‘zgaruvchilar bir-biridan “>>” belgisi bilan ajratib ko‘rsatiladi. Masalan:

cin >>x1>>x2.

Cin buyrug‘ini bajargan kompyuter ishdan to‘xtaydi va ro‘yxatda ko‘rsatilgan barcha o‘zgaruvchilar uchun qiymat kiritilishini kutadi. Klaviaturadan kiritilayotgan ma’lumotlar bir-biridan bo‘sh joy belgisi bilan ajratilishi lozim. Bu ma’lumotlar tartib raqamlariga qarab mos ravishda ro‘yxatda ko‘rsatilgan o‘zgaruvchilarga qiymat qilib beriladi. Boshqacha aytganda, birinchi kiritilgan ma’lumot ro‘yxatda turgan birinchi o‘zgaruvchiga, ikkinchi ma’lumot ikkinchisiga va hokazo tartibda beriladi. Yuqoridagi buyruq uchun klaviaturadan quyidagi ma’lumotlar kiritilgan bo‘lsin:

18.4 15

U holda *x1* o‘zgaruvchiga 2.34, *x2* ga esa 15 qiymati beriladi va dasturdagi navbatdagi buyruqlar o‘zgaruvchilarning aynan ana shu qiymatlari uchun bajariladi.

Qiymat olayotgan o‘zgaruvchi bilan unga berilayotgan qiymat bir xil tipga mansub bo‘lishi lozim.

Real tipida ma’lumot kiritilayotganda butun sonlarni ham kiritishga ruxsat beriladi, chunki butun sonlar haqiqiy sonlar to‘plamiga kiradi (bu holda kompilyator kiritilgan 10 sonini 10.00 tarzida qabul qiladi).

Klaviatura orqali kiritilgan ma’lumotlar soni *cin* operatorida berilgan ro‘yxatdagi o‘zgaruvchilar sonidan kam bo‘lmasligi lozim. Aks holda, ro‘yxatda ko‘rsatilgan qaysidir o‘zgaruvchilar qiymat olmagani sababli, navbatdagi buyruqlar bajarilmay turaveradi.

Agar kiritilgan ma’lumotlar soni *cin* operatori bilan ko‘rsatilgan o‘zgaruvchilar sonidan ko‘p bo‘lsa, buning zarari yo‘q. Chunki ortiqcha qiymatlar yoki navbatdagi *cin* da ko‘rsatilgan o‘zgaruvchilarga qiymat qilib beriladi yoki tashlab yuboriladi, ya’ni e’tiborga olinmaydi. Masalan, bitta dasturda

cin (a, b, c); cin (x, y);

operatorlariga javoban klaviaturadan

2.3 -1.5 2.4 22 -0.05 4.125

ma’lumotlari kiritilgan bo‘lsin. U holda *a* o‘zgaruvchi 2.3, *b* – 1.5, *c* esa 2.4 qiymatlarini qabul qiladi, navbatdagi 22 va -0.05 qiymatlari mos ravishda *x* va *y* o‘zgaruvchilarga beriladi. Ortiqcha kiritilgan 4.125 ma’lumotidan kompyuter foydalanmaydi, ya’ni tashlab yuboradi.

Bu ma’lumotlarni e’tiborga olib, 3-masala kodini quyidagicha yozish mumkin:

```
 # include <iostream.h>
int main()
{  float L, R, pi, S;
pi=3.14;
cin>> L;
R=L/(2*pi);
S=pi*R*R;
cout<<”S=”<<S;
   return 0;
   }
```

Takrorlash uchun savol va topshiriqlar

1. C++ tilidagi dasturning umumiy ko‘rinishi qanday?
2. *Cin* va *cout* operatorlarining vazifasini ayting.
3. *Cout* operatori yordamida hisoblashlarni bajarish mumkinmi?
4. Quyidagi masalalar uchun algoritm va dastur ishlab chiqing:

a) to‘g‘ri burchakli uchburchakning katetlari berilgan bo‘lsin. Uning gipotenuzasi, perimetri va yuzini hisoblang;

b) asosining tomoni *a*, balandligi *h* bo‘lgan oltiburchakli to‘g‘ri

prizmaning to‘la sirtini hisoblang;

c) silindr yon sirtining yoyilmasi tomoni *a* ga teng kvadratdan iborat bo‘lsa, uning hajmi topilsin;

d) balandligi, o‘tkir burchagi va tomonlaridan biri berilgan teng yonli trapetsiya yuzini toping;

e) asosining tomoni *a* va yon qirrasi *b* bo‘lgan muntazam oltiburchakli piramidaning to‘la sirtini toping;

f) $y=x^{29}$ ni darajaga ko‘tarish amalidan foydalanmagan holda eng kam ko‘paytirishlar yordamida hisoblang.

4-§. O‘TISH, TARMOQLANISH VA TANLASH BUYRUQLARI

4.1. Mantiqiy ifodalar

Inson hayoti davomida doimo qandaydir masalalarni hal qilishga harakat qiladi. Shu masalalarni yechish jarayonida u mumkin bo‘lgan turli mulohazalar va ularning oqibatlarini hisobga olgan holda bu ishga qo‘l uradi. Masalan, ishga otlanayotgan kishi ertalab uydan chiqishidan oldin “hozir kuchli yomg‘ir yog‘moqda” mulohazasini xayolan tahlil qiladi va soyabonni o‘zi bilan olish-olmaslik masalasini hal qiladi. “Hozir harorat 20° dan yuqori” mulohazasining natijasi esa uning kiyadigan kiyimlarini belgilab beradi. Bunday vaziyatlarda mumkin bo‘lgan amallardan qaysi birini bajarish qaralayotgan mulohaza (mantiqiy ifoda) qiymatiga bog‘liq bo‘ladi.

Mantiqiy ifoda qiymati “rost” yoki “yolg‘on” bo‘lishi mumkin bo‘lgan ilmiy-nazariy va hayotiy mulohazalar, qisqa qilib aytganda, shartlardan iborat bo‘ladi.

Mantiqiy mulohaza natijalarini EHM yordamida tahlil qilish uchun ularni C++ tilida maxsus usulda ifodalash lozim. Buning uchun munosabat belgilaridan foydalaniladi. Bu belgilarni yozish tartibi quyidagicha:

Munosabat	C++ da	Oddiy yozuv	C++ da
=	= =	$x^2 = 10$	x*x==10
≠	!=	$x \neq 5$	x != 5
>	>	$x^2 - 3x > 0$	x*x-3*x>0
≥	>=	$5x \geq 7$	5*x>=7
<	<	$x^2 - 4ac < 0$	x*x-4*a*c<0
≤	<=	$ab \leq cd$	a*b<=c*d

Ikki va undan ortiq shartlardan iborat murakkab mantiqiy mulohazalarni tahlil qilishda mantiqiy “ha” (bir nechta mulohazalarning bir vaqtda o‘rinli bo‘lishi), “yoki” (bir nechta mulohazalardan kamida bittasining o‘rinli bo‘lishi) hamda “emas” (mulohazaning teskarisi) kabi amallardan foydalanish mumkin.

Bunday mulohazalarning qiymatlarini aniqlash uchun hosil qilingan murakkab mantiqiy ifodalardagi shartlar qavslar ichida ko‘rsatilishi lozim.

1. (“Hozir yomg‘ir yog‘moqda”) va (“harorat 20° dan past”).
2. (“Hozir yomg‘ir yog‘moqda”) yoki (“harorat 20° dan past”).
3. Emas (“Hozir yomg‘ir yog‘moqda”).

Bu mantiqiy ifodalarning birinchisi faqat har ikki shart o‘rinli bo‘lgan holdagina “rost”, qolgan hamma hollarda esa “yolg‘on” qiymatni oladi. Ikkinchi mantiqiy ifoda esa ikki shartdan kamida bittasi “rost” bo‘lganda “rost” qiymatini oladi. 3-mulohazadagi “emas yoki inkor” amali qavs ichidagi mulohaza natijasini teskarisiga almashtiradi.

A va B mulohazalar berilgan bo‘lsin. Ular uchun mantiqiy amallar C++ tilida quyidagicha yoziladi:

Munosabat	C++ da	Oddiy yozuv	C++ da
and	&&	A va B	A && B

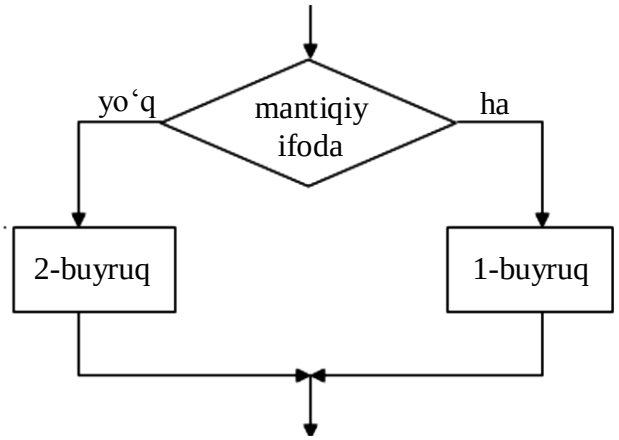
or	||	A yoki B	A || B
not	!	A ning inkori	! A

4.2. Tarmoqlanish buyrug‘i

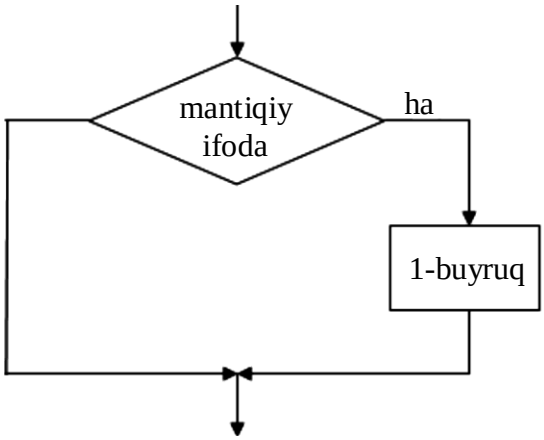
Ko‘pincha masalaning yechimini topish qandaydir mantiqiy amalga bog‘liq bo‘lib qolishi mumkin. Masalan, diskriminantning noldan katta yoki kichik bo‘lishi kvadrat tenglama yechimlarini aniqlashda muhim omil hisoblanadi. Bunday masalalar uchun dastur ishlab chiqishda tarmoqlanish buyrug‘idan foydalaniladi. Uning umumiy ko‘rinishi quyidagicha:

if (mantiqiy ifoda) {1-buyruqlar } ***else*** {2-buyruqlar}

Bu buyruqni bajarishda EHM dastlab “mantiqiy ifoda” qiymatini aniqlaydi. Agar u “rost” bo‘lsa, 1-buyruqlar ketma-ketligi bajariladi, 2-buyruqlar esa bajarilmaydi. Aks holda 2-buyruqlarni bajarib, 1-buyruqlarni bajarmaydi (4.1-rasmdagi blok-sxemada tarmoqlanish buyrug‘ining bajarilish tartibi ko‘rsatilgan.) So’ngra kompyuter ***if*** dan keyin ko’rsatilgan buyruqni bajarishga o‘tadi.



4.1-rasm



4.2-rasm

Tarmoqlanish buyrug‘idan foydalanib kvadrat tenglama yechimining mavjudligi quyidagicha yoziladi:

if *(D>=0)* **{**cout << “yechim mavjud”**}** ***else*** **{**cout << “yechim yo‘q”**}**

Vaziyatga ko‘ra tarmoqlanish buyrug‘ining qisqartirilgan varian-tidan ham foydalanish mumkin. Bu buyruq umumiy ko‘rinishda quyi-dagicha yoziladi:

if (mantiqiy ifoda) {buyruqlar}

Kvadrat qavslar orasida ko‘rsatilgan buyruqlar mantiqiy ifoda “rost” qiymat olgan holdagina bajariladi. Aks holda hech bir amal bajarilmaydi va ***if*** dan keyingi buyruqqa o‘tiladi. (Bu buyruqqa mos keladigan blok-sxema 4.2-rasmda keltirilgan.)

1-masala. Xodim bir oyda *N* soat ishlaydi va har bir ish soati uchun *K* so‘m ish haqi oladi. U 200 soatdan ortiq ishlagan har bir soati uchun 2 baravar ko‘p ish haqi tayinlangan. Xodimning oylik ish haqini aniqlang.

Yechish g‘oyasi. Xodimning ish haqini ***ish_haqi***, ishlagan soatlarini *N* bilan belgilaylik. U holda uning ish haqi

$$ish\_haqi = \begin{cases} K * N, \ \ agar \ \ N <= 200 \\ K * 200 + (N - 200) * K * 2, \ agar \ N > 200 \end{cases}$$

formula yordamida hisoblanadi. Bu formulani e’tiborga olib, dastur kodini quyidagicha yozish mumkin:

```
# include <iostream.h>
int main()
{
int N, K;
unsigned int ish_haqi;
cout <<”Xodimning 1 soatlik ish haqi=\n”;
cin >>K;
cout <<”Uning bir oyda ishlagan soatlari=\n”;
cin >>N;
if (N<=200) ish_haqi=N*K;
else ish_haqi = 200*K+ (N−200)*K*2;
 cout <<ish_haqi<<”\n”;
return 0;
```

```
}
```

Ushbu kod uchun kompyuter quyidagi natijani beradi:

```
Xodimning 1 soatlik ish haqi= 120
Uning bir oyda ishlagan soatlari= 120
14400
Xodimning 1 soatlik ish haqi=120
Uning bir oyda ishlagan soatlari=240
33600
```

Bu kodda tarmoqlanish buyrug‘ining to‘liq varianti qo‘llanilgan. Uning qisqartirilgan variantidan foydalangan kod quyidagicha yoziladi:

```
# include <iostream.h>
int main()
{
int N, K;
unsigned int ish_haqi;
cout <<”Xodimning 1 soatlik ish haqi=\n”;
cin >>K;
cout <<”Uning bir oyda ishlagan soatlari=\n”;
cin >>N;
ish_haqi=N*K ;
if (N>200) ish_haqi=200*K+(N−200)*K*2;
 cout <<ish_haqi<<”\n”;
return 0;
}
```

Bu kodni quyidagicha izohlash mumkin: ish haqi hamma uchun bir xil, ya’ni $ish\_haqi := haq * ish\_soati$ formula bilan hisoblanadi. Qo‘shimcha ish haqini esa faqat bir oyda 200 soatdan ortiq ishlagan xodimlar oladi.

Ikki va undan ortiq mulohazalardan && (*and),* || *(or)* va ! (*not)* mantiqiy amallaridan foydalanib ifodalar tuzish mumkin. Bunda har bir mulohaza qavslar ichida ko‘rsatilishi shart. Quyidagi misollarga

e’tibor bering:

2-misol: *X* sonining [2, 5] oraliqqa tegishli ekanligini aniqlang.

Bu masala uchun ortiqcha izoh zarur bo‘lmagani uchun, uning kodini keltirish bilan cheklanamiz:

```
# include <iostream.h>;
int main()
{
float x;
cin >>x;
if ((x>=2)&&(x<=5)) cout <<”tegishli”;
                else cout <<”tegishli emas\n”;
return 0;
}
```

4.3. Tanlash operatori – switch

Ayrim masalalarni yechish jarayonida mavjud ko‘plab variantlardan birini tanlab olish uchun ichma-ich tarmoqlanish, ya’ni bitta ***if*** operatori ichida bir nechta ***if*** operatorlarini qo‘llashga to‘g‘ri keladi. Bunday holat dastur yozishni ham, uni tushunishni ham murakkablashtirib yuboradi. Bunday muammoning oldini olish uchun C++ tilida *switch* operatori qo‘llaniladi. Bu operator ***if*** dan qiymatga ko’ra mumkin bo’lgan variantlardan birini tanlab olish imkoni bilan farqlanadi. Natijada dasturni o’qish va tushunish osonlashadi. *Switch* operatori umumiy ko‘rinishda quyidagicha yoziladi:

```
switch (kalit)
{
case 1 – qiymat: 1 – operatorlar ketma – ketligi;  break;
case 2 – qiymat: 2 – operatorlar ketma – ketligi;  break;
 …..
case N – qiymat: N – operatorlar ketma – ketligi;   break;
default: N+1  - operatorlar ketma – ketligi;
}
```

Switch operatoridagi kalit o‘rnida natural son, qiymati natural son bo‘lgan arifmetik ifoda, belgilar kelishi mumkin.

Operator quyidagicha bajariladi: dastlab kalitning qiymati aniqlanadi. So‘ngra ***case*** da ko‘rsatilgan ana shunday qiymatga mos operatorlar ketma-ketligi bajariladi. Har bir opreratordan keyin ko‘rsatilgan *break* ko‘rsatmasi boshqaruvni ***switch*** dan keyingi operatorga uzatadi. Agar kalitning qiymati *case* dagi qiymatlarning birortasiga teng bo‘lmasa, *default* da ko‘rsatilgan N+1-chi operatorlar ketma-ketligi bajariladi.

Masala. Sportchi musobaqada *N*-o‘rinni olgan bo‘lsin. U qanday medal olgan?

Masalaning yechish g‘oyasi yetarlicha sodda bo‘lgani uchun uni keltirmaymiz. Dastur kodini tahlil qilish unchalik qiyin emas.

```
# include <iostream.h>
void main()
{ int N;
cout<<”Sportchi nechanchi o‘rinni olgan?”;
cin>>N;
      switch (N)
      {
      case 1 : cout<<”Oltin”; break;
      case 2 : cout<<”Kumush”; break;
      case 3 : cout<<”Bronza”; break;
      default
      cout<<”Sportchi medal olmagan”; break;
      }
 }
```

Takrorlash uchun savol va topshiriqlar

1. Mantiqiy ifoda nima va u qanday qiymatlarni olishi mumkin?
2. Murakkab mantiqiy ifodalar qanday tuziladi?
3. Tarmoqlanish buyrug‘ining to‘liq va to‘liqsiz ko‘rinishlarini

ayting.

4. Tamg‘a nima?

5. Tanlash va tamg‘a buyruqlaridan qachon foydalanish mumkin?

6. Quyidagi masalalar uchun dastur ishlab chiqing.

a) *x* va *y* haqiqiy sonlari berilgan. Ularning kattasini toping.

b) Uchta haqiqiy *x*, *y* va *z* sonlari berilgan bo‘lsin. Ularning eng kichigini toping.

c) Berilgan *N* natural son juftmi yoki toqmi?

d) *A* va *B* natural sonlari berilgan bo‘lsin. *A* soni *B* ning bo‘luvchisi bo‘la oladimi?

e) $Ax^3 + Bx = 0$ tenglamaning yechimlari sonini aniqlang.

f) Hafta kunining nomeri berilgan bo‘lsin. Bu nomerga haftaning qaysi kuni mos keladi?

5-§. SIKLLARNI TASHKIL QILISH

5.1. WHILE operatori

Ko‘plab masalalarni yechish jarayonida ayrim amallar ketma-ketligini takror va takror ko‘rsatishga va demak bajarishga to‘g‘ri keladi. 4-§ dagi 1-masalani esga oling. U yerda bir nechta buyruqlarni 30 ta xodimi bor korxona uchun 30 marta bajarishga to‘g‘ri keladi.

Bitta dastur tarkibida bir necha marta bajariladigan buyruqlar ketma-ketligini sikl deb atash qabul qilingan.

C++ tilida sikllarni tashkil qilishning bir necha usullari mavjud. Ulardan biri *while* operatoridir.

While (inglizcha – “toki”) operatori umumiy ko‘rinishda quyidagicha yoziladi:

While (mantiqiy ifoda *) {1 – buyruq ;... n – buyruq ;};*

Agar sikl bitta buyruqdan iborat bo‘lsa, operatorni qisqa ko‘rinishda ham yozish mumkin:

*While (*mantiqiy ifoda*) buyruq ;*

While operatori quyidagicha tartibda bajariladi: dastlab mantiqiy ifodaning qiymati hisoblanadi. Agar uning qiymati “rost” bo‘lsa, sikldagi buyruqlar ketma-ketligi bir marta bajariladi, so‘ngra yana mantiqiy ifodaning qiymati aniqlanadi. Agar qiymat “rost” bo‘lsa, sikl yana bir marta bajariladi va hokazo. Bu jarayon toki mantiqiy ifodaning qiymati “yolg‘on” bo‘lguncha davom etadi. Agar uning qiymati birinchi marta tekshirilganda “yolg‘on” bo‘lsa, sikldagi buyruqlar ketma-ketligi bir marta ham bajarilmaydi.

Quyidagi kod ko‘rsatilgan marta salom berilgandan keyin xayrlashish holatini namoyish qiladi.

```
# include <iostream.h>
int main()
{
int marta;
cout<<”Necha marta salomlashamiz\n”;
cin >> marta;
while (marta>0) {
cout<<”Salom\n”;
marta=marta – 1;
}
cout <<”Hayr, Janob\n”;
return 0;
}
```

EHM bu dastur uchun quyidagi natijalarni beradi:

```
Necha marta salomlashamiz 3
Salom
Salom
Salom
Hayr, Janob!
Necha marta salomlashamiz? 0
```

Hayr, Janob

While operatoridan takrorlanishlar soni oldindan noma'lum bo'lgan hollarda foydalanish maqsadga muvofiq hisoblanadi.

1-masala. *A* va *B* natural sonlar berilgan bo'lsin. Ularning eng katta umumiy bo'luvchisini toping.

Yechish g'oyasi. Bu masala "Evklid algoritmi" asosida hal qilinadi. Unga ko'ra, to *A* va *B* sonlari bir-biriga teng bo'lib qolmaguncha, kattasidan kichigi ayriladi va ayirma shu sonlardan kattasini ifodalab turgan o'zgaruvchi nomi bilan belgilab qo'yiladi. Bu jarayon *A*=*B* bo'lganda to'xtaydi. Yechim sifatida *A* yoki *B* ni olish mumkin.

```
# include <iostream.h>
int main()
{
int a, b;
cout<<"a va b butun sonlarni kiriting\n";
cin >>a>>b;
cout <<a<<b<<"\n";
while (a != b) if (a>b) a=a-b; else b=b-a;
cout<<"EKUB="<<a<<"\n";
return 0;
}
```

Ushbu dasturni ***a***=60 va ***b***=36 uchun bajargan kompyuter

$$EKUB = 12$$

ko'rinishidagi natijani ekranga chiqaradi.

Eslatma. Mantiqiy ifodaning qiymatiga ko'ra *while* sikli bir marta ham bajarilmasligi mumkin.

5.2. FOR operatori

Ko'plab sikllarni bitta o'zgaruvchining *m, m+1, ..., N* bo'lgan qiymatlari uchun bajarishga to'g'ri kelib qoladi. Ana shunday jarayonlarni ixcham qilib yozish maqsadida *for* sikl operatori kiritilgan. U

umumiy ko‘rinishda quyidagicha yoziladi:

for (initsializator; shart; o‘zgarish qadami) {buyruqlar;}

Agar siklda takroran bajarilishi talab qilingan buyruq bitta bo‘lsa, ***for*** operatorini quyidagicha yozish mumkin:

for (initsializator; shart; o‘zgarish qadami) buyruq;

Ko‘rinib turibdiki, bu operator uch blokdan iborat va bu bloklarning ixtiyoriy biri bo‘sh bo‘lishi mumkin. Bu bloklar quyidagi ma’noni anglatadi:

initsializator – takrorlash jarayonini boshqaradigan o‘zgaruvchining boshlang‘ich qiymati;

shart – siklning bajarish yoki bajarilmasligini belgilab beruvchi shart;

o‘zgarish qadami – siklni boshqaruvchi o‘zgaruvchining o‘zgarish qadami.

Masalan:

for (x=x1; x<=x2; ++x) cout <<”salom\n”;

For buyrug‘i birinchi marta bajarilganda, *x* o‘zgaruvchiga *x1* qiymat beriladi. So‘ngra *x* ning ana shu qiymati uchun sikl bir marta bajariladi. Keyin *x* o‘zgaruvchining qiymati o‘zgarish qadamiga mos ravishda o‘zgartiriladi va *x* ning yangi qiymati uchun *x<=x2* shartning qiymati aniqlanadi. Agar u “rost” bo‘lsa, sikl yana bir marta bajariladi va hokazo. Bu jarayon shart “yolg‘on” bo‘lganda to‘xtaydi.

For sikli hech bo‘lmaganda bir marta bajariladi.

2-masala. $1+2+3+...+1000$ yig‘indini toping.

Yechish g‘oyasi. Yig‘indini hisoblab borish uchun *S* o‘zgaruvchi kiritiladi. So‘ngra *I* ning 1 dan 1000 gacha bo‘lgan qiymatlari uchun $S = S + i$ ifoda qiymatini hisoblanadi.

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

```
long int S; int i;
S=0;
for (i=1; i<=1000; ++i) S=S+i;
cout<<S<<"\n";
return 0;
}
```

EHM bu dastur uchun quyidagi natijani beradi:

S:=500500

3-masala. *N* natural soni berilgan bo‘lsin. Uning barcha bo‘luvchilarini kamayish tartibida yozing.

Yechish g‘oyasi. Ma’lumki, har qanday natural son bo‘luvchilari o‘zining yarmidan kichik bo‘ladi. Shu sababli berilgan sonning mumkin bo‘lgan bo‘luvchilarini [*N*/2, 1] oraliqdan izlaymiz.

Bu go‘yaga mos keluvchi kod quyidagicha yoziladi.

```
# include <iostream.h>
int main()
{
int N, i, j;
cin>>N;
j=N / 2;
cout<<"=="<<N<<" , n";
for (i=j; i>=1; −−i)
{if (N % i==0) cout<<i<<" , ";}
return 0;
}
```

Bu dastur *N*=24 uchun quyidagi natijani beradi:

24 , 12, 6, 4, 3, 2, 1

Quyida keltirilgan kod tarkibidagi sikl har gal bajarilganda, *k* ning qiymati 2 ga ortib boradi va takrorlash jarayoni *k*=12 bo‘lganda to‘xtaydi.

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

```
int main()
{
int k, n;
k=1;
while (k != 12) {cout<<k<<"\n"; k+=2;}
return 0;
}
```

Lekin, ushbu dasturdagi *k*=2 buyrug‘i *k*=1 bilan almashtirilsa, *k* ning qiymati hech qachon 12 ga teng bo‘lmaydi. Buning oqibatida dasturning bajarilish jarayoni hech qachon tugamaydi, ya’ni “cheksiz sikl” hodisasi ro‘y beradi.

Ayrim hollarda siklning bajarilish jarayonini to‘liq tugatmay turib, to‘xtatishga to‘g‘ri keladi. Bu amalni *break* buyrug‘i yordamida amalga oshirish mumkin. Agar dasturda *break* buyrug‘i uchrab qolsa, EHM faqat sikl jarayonini emas, balki navbatdagi buyruqlarni ham bajarmay, o‘z ishini to‘xtatadi.

Sikl buyrug‘i o‘z ichiga ko‘plab buyruqlarni olishi mumkin. Shu jumladan bitta sikl ichida boshqa bir sikl ham kelishi mumkin. Bunday sikllar ichma-ich joylashgan sikllar deb ataladi. Bunda avval boshlangan sikl tashqi, keyingisi esa ichki deb yuritiladi.

Tashqi siklni boshqarayotgan o‘zgaruvchining bir marta o‘zgarishi uchun ichki sikl to‘la bir marta bajariladi (soatning soat va minut ko‘rsatkichlarini esga oling. Unda soat milining bir marta o‘zgarishiga minut mili 0 dan 59 gacha bir marta to‘la aylanib chiqadi. Bu yerda soat milini tashqi, minut milini esa ichki siklning boshqaradigan o‘zgaruvchilar sifatida qarash mumkin.) Ichki va tashqi sikllar har gal bu o‘zgaruvchilarning joriy vaqtdagi qiymatlari uchun bajariladi.

Quyida keltirilayotgan kod Dekart ko‘paytmalar jadvalini ekranga chiqaradi:

```
# include <iostream.h>
int main()
{
int i, j, s;
```

```
for (i=1; i<=10; i++) {
for (j=1; j<=10; j++)
{
 s=i*j;
cout <<s<<"  “;}
cout <<"\n";
}
return 0;
}
```

Ushbu kodni tekshirib ko‘rish o‘zingizga havola.

Tekshirish uchun savol va topshiriqlar

1. Sikl deganda nimani tushunasiz?
2. *While* operatori bilan sikllarni qanday tashkil qilish mumkin?
3. *For* operatori haqida nimalarni bilasiz?
4. Cheksiz sikl qanday hollarda yuz beradi?
5. Quyidagi masalalar uchun dastur ishlab chiqing.

a) Quyidagi ifodalarning qiymatlarini hisoblang:

a.1	$\prod_{k=1}^{n} \frac{(1-k)^2+1}{((k-1)!+1)^2}$	*a*.2	$\sum_{k=0}^{n} (-1)^k \frac{k+1}{(2k)!}$
a.3	$\sum_{k=1}^{\infty} \frac{(-1)^k x^{4k+3}}{(2k+1)!\,(4k+3)}$	*a*.4	$\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{(2k+1)!} \left(\frac{x}{3}\right)^{4k+3}$

b) A va *B* natural sonlari berilgan bo‘lsin. Ularning eng kichik umumiy ko‘paytuvchisini toping.

c) Fibonachchi sonlari ketma-ketligi $y_0=0$, $y_1=1$, $y_i=y_{i-1}+y_{i-2}$ formulalar yordamida topiladi. *K* (*K*>0) sonidan katta bo‘lgan birinchi hadini toping.

d) N natural soni berilgan bo‘lsa, *N* !! ni hisoblang.

f) N natural soni berilgan bo‘lsin. $à^2 + b^2 = N^2$ shartni qanoat-

lantiruvchi barcha *a* va *b* larni aniqlang.

6-§. MASSIVLAR

6.1. Massivlar va ulardan foydalanish

Tashkiliy, ishlab chiqarish, iqtisodiyotga doir koʻplab masalalar uchun dastur yozishda bir xil tipdagi va katta sondagi maʼlumotlar bilan ishlashga toʻgʻri keladi. Masalan, abituriyentlarning familiyalari roʻyxati, kirish imtihonida ularning olgan baholari, bitta tashkilot xodimlarining oylik maoshlari, yilning har bir kunidagi oʻrtacha harorat va h.k.

Bunday maʼlumotlarni C++ tilida toʻgʻridan-toʻgʻri hisobga olish mushkul masala. Mazmuni va tipi bir xil boʻlgan maʼlumotlardan jadvallar tashkil qilish bunday muammolarni oson hal qilish usullaridan biri hisoblanadi. Jadvalga elementlarni chiziqli va toʻgʻri toʻrtburchakli koʻrinishda joylash mumkin.

Elementlari bitta satrga yoki ustunga joylashtirilgan jadvallar chiziqli (yoki bir oʻlchovli) massivlar deb ataladi.

Elementlari bir nechta satr va ustunlarga joylashtirilgan jadvallar toʻgʻri toʻrtburchakli (yoki ikki bir oʻlchovli) massivlar deb ataladi.

Abituriyentlar roʻyxatini bir ustunli, bitta talabaning olgan baholarini bitta satrli, guruhdagi talabalarning barcha fanlardan olgan baholarini esa toʻgʻri toʻrtburchakli jadvalga joylash mumkin. Masalan:

Abdullayeva B
Valiyeva A
Komilov A
Murodov K
Soliyev T
Qodirova M

78	82	75	84	87	74	79

79	73	78	81
67	62	71	76
87	86	87	90
74	74	77	82
72	75	87	86
78	79	86	79

C++ tilida chiziqli jadvallar bilan ishlash uchun bir oʻlchovli va

to‘g‘ri to‘rtburchak shaklidagi jadvallar bilan ishlash uchun ikki o‘lchovli massivlar ko‘zda tutilgan. Shuningdek, C++ da o‘lchovi ikkidan katta bo‘lgan massivlar bilan ham ishlash imkoniyati ko‘zda tutilgan.

Massivdagi ma’lumotlar uning elementlari deb ataladi. Biror elementga murojaat qilish uchun uning massivda tutgan o‘rnidan (indeksidan) foydalaniladi.

Massivlarning nomi ikki qismdan iborat bo‘lgan hamda bir xil tipdagi o‘zgaruvchilar guruhi (jadvali) deb qaraladi. Nomning birinchi qismi – massiv nomi bu guruhdagi hamma ma’lumotlar uchun bir xil, ikkinchi qismi esa elementlarning massivda tutgan o‘rnini bildiradi.

Bir o‘lchovli massivlar umumiy ko‘rinishda quyidagicha e’lon qilinadi:

tip *massiv nomi [A];*

bu yerda *A* – massivdagi elementlar sonini bildiradi. Shu bilan birga bu son massiv elementlari indekslarining o‘zgarish diapazonini ham anglatadi.

Ikki o‘lchovli massivlar esa

tip *massiv nomi [A][B];*

ko‘rinishda e’lon qilinadi. Bu yerda *A* soni massiv satrlari sonini (diapazonini), *B* esa ustunlar sonini (diapazonini) anglatadi. Bunday usulda e’lon qilingan massiv elementlarining soni $A \times B$ ga teng bo‘ladi. Masalan :

float X[100] ;
int baho[6]][4];

e’loni yordamida 100 ta elementli *X* hamda elementlari soni 24 ta bo‘lgan 6 ta satrli va 4 ta ustunli baho massivlari e’lon qilinmoqda.

E’lon qilingan har bir massiv uchun ma’lum bir hajmda xotira yacheykalari (o‘zgaruvchi tipiga bog‘liq ravishda) ajratiladi va kompyuter bu yacheyka adreslarini xotirasida saqlab turadi. Massivlar uchun kompyuter faqat birinchi element turgan adresni “eslab qoladi”, qolgan elementlarning xotirada tutgan o‘rinlarini ana shu adresga

massiv tipini hisobga olgan holda ma’lum bir sonni qo‘shish orqali hosil qiladi.

Biror elementiga murojaat qilish massiv nomi va kvadrat qavs ichida shu elementning massivdagi o‘rnini (indeksini) ko‘rsatish orqali (masalan: *X*[4], *baho*[3][2] kabi) amalga oshiriladi.

Bir o‘lchovli massiv elementlarining boshlang‘ich indeksi 0 ga, oxirgi indeksi esa massiv elementlari sonidan bittaga kam bo‘ladi. Ikki o‘lchovli massivlar uchun esa satr va ustunlarning oxirgi indekslari mos ravishda satr va ustun sonlaridan bittaga kam bo‘ladi

Massiv elementining indekslari kvadrat qavslar ichida ko‘rsatilgan sonlar orasidan chetga chiqmasligi kerak. Masalan, yuqorida e’lon qilingan massivlar uchun *X*[100] yoki *baho*[6][3] kabi elementlar mavjud emas. Chunki bu elementlarning indekslari e’lon qilingan diapazonga kirmaydi.

Massiv elementlarining indekslari o‘rnida ixtiyoriy tartiblangan tipdagi (masalan, integer) va qiymati indekslar diapazonidan chetga chiqmaydigan turli ifodalar ham kelishi mumkin. R[12*8-3] bilan R[93] yozuvlari bitta elementni anglatadi.

Ixtiyoriy vektor, guruhdagi talabalar ro‘yxati, talabalarning bitta imtihondan olgan baholaridan tuzilgan jadvallar bir o‘lchovli massivlarga misol bo‘la oladi. Matritsalarni esa ikki o‘lchovli massiv sifatida qabul qilish mumkin. Masalan,

A_1	B_1 B_2 B_3 B_4 B_5 B_6	C_{11} C_{12} C_{13} C_{14}
A_2		C_{21} C_{22} C_{23} C_{24}
A_3		C_{31} C_{32} C_{33} C_{34}
A_4		C_{41} C_{42} C_{43} C_{44}

kabi jadvallarni quyidagicha e’lon qilish mumkin:

float A[4]; int B[6]; int C[4][4];

Elementlari oldindan ma’lum bo‘lsa massivlarni uni e’lon qilishda elementlar sonini ko‘rsatmaslik ham mumkin. Bu holda massiv o‘lchami uning elementlari soniga ko‘ra to‘g‘ridan-to‘g‘ri aniqlanadi. Masalan,

int A[]={1, 3, 5, 7};

ko‘rsatmasi elementlari mos ravishda {1, 3, 5, 7} sonlarga teng bo‘lgan to‘rtta elementli *A* massivni e’lon qilishga ekvivalent hisoblanadi.

Massiv elementlari ustida arifmetik amallar oddiy o‘zgaruvchilar bilan bir xil ko‘rinishda bajariladi:

*R[12]=10*2 ; X[12]:=R[12]/2*

1-masala: *A*[1:l00] butun sonli massivda qiymati 7 ga teng elementlar sonini aniqlang.

Yechish g‘oyasi. Dastlab 7 ga teng bo‘lgan elementlarni sanash uchun *s*=0 o‘zgaruvchi tanlanadi. So‘ngra massiv elementlarini ketma-ket kiritib, 7 soni bilan taqqoslanadi. Agar bu element 7 ga teng bo‘lsa, *s* ning qiymati 1 ga orttiriladi. Keyin navbatdagi elementga o‘tiladi. Bu jarayon 100 ta elementlarning har biri uchun takrorlanadi.

Qo‘yilgan masala uchun dasturni quyidagicha yozish mumkin:

```
# include <iostream.h>
int main()
{
int a[100]; int i, s;
s=0;
for (i=0; i<=99; i++)
{
cout<< “massivning “<<i+1<<”- elementini kiriting”;
cin>>a[i];
if (a[i]==7) s=s+1;
}
cout<<“Bunday elementlar soni=”<<s<<“\n”;
return 0;
}
```

Ushbu kodni ishga tushiramiz.

massivning 1-elementini kiriting 7

massivning 2-elementini kiriting 4
massivning 3-elementini kiriting 3
massivning 4-elementini kiriting 7
massivning 5-elementini kiriting 8
massivning 6-elementini kiriting 6
massivning 7-elementini kiriting 5
massivning 8-elementini kiriting 7
massivning 9-elementini kiriting 8
massivning 10-elementini kiriting 9
Bunday elementlar soni= 3

2-masala. 10 ta haqiqiy elementli massivdagi eng katta va eng kichik elementlarni toping.

Yechish g‘oyasi. Dastlab massivning barcha elementlari kiritiladi. So‘ngra ularning birinchisini izlangan element, ya’ni eng kattasi va eng kichigi deb faraz qilinadi. Qolgan elementlar uchun bu farazning to‘g‘ri yoki noto‘g‘riligi tekshiriladi. Agap eng katta (eng kichik) degan elementdan ham kattaroq (kichikroq) element topilib qolsa, izlangan element sifatida u qabul qilinadi va navbatdagi tekshirishlarni ana shu element uchun davom ettiriladi. Bu masalaning dasturi quyidagicha yoziladi:

```
# include <iostream.h>
int main()
{ float a[10]; float max, min; int i;
for (i=0;i<=9;i++)
{
cout<<“massivning “<<i+1<<”- elementini kiriting”;
cin>>a[i]; cout<<”\n”;
}
max=a[0]; min=a[0];
for (i=1; i<=9; i++) {
if (a[i]>max) max=a[i];
if (a[i]<min) min=a[i];
```

```
}
cout<<“Eng katta element=”<<max<<“\n”;
cout<<“Eng kichik element=”<<min<<“\n”;
return 0;
}
```

Kodni kompyuterda bajaramiz.

```
massivning 1-elementini kiriting 3
massivning 2-elementini kiriting 4
massivning 3-elementini kiriting 5
massivning 4-elementini kiriting 3
massivning 5-elementini kiriting 5
massivning 6-elementini kiriting 9
massivning 7-elementini kiriting 1
massivning 8-elementini kiriting -3
massivning 9-elementini kiriting 7
massivning 10-elementini kiriting 4
Eng katta  element=9
Eng kichik element=-3
```

Matnli ma’lumotlarni ***char*** tipidagi elementlar massivi sifatida qabul qilish mumkin. Masalan:

char R [18]

yozuvi *R* massiv *char* tipidagi 18 ta elementdan iborat ekanligini bildiradi. *R* ning biror belgisiga murojaat qilish zarur bo‘lsa, uning turgan o‘rni ko‘rsatiladi. Masalan, K:= ”MATEMATIKA-GEOMETRIYA” matni uchun *R*[9]=“A” va *R*[16]=“T” bo‘ladi.

Eslatma. *Char* tipidagi massivlarga belgilangan diapazondan uzunroq matnlarni qiymat qilib berish mumkin emas, aks holda uning ortiqcha qismi tashlab yuboriladi.

3-masala. 18 tagacha belgisi bo‘lgan *W* so‘zini teskarisidan o‘qish dasturini yozing.

Yechish g‘oyasi. Dastlab *W* so‘zi kiritiladi. So‘ngra hech qanday belgisi bo‘lmagan, ya’ni bo‘sh *s* o‘zgaruvchi olinadi. Uning chap

tomoniga berilgan so‘zning birinchi belgisidan boshlab, hamma belgilarni bitta-bittadan keltirib, yonma-yon yoziladi.

```
# include <iostream.h>
int main()
{
char w[10], s[10]; int i;
cin>>w;
for (i=0; i<=9; i++) s[9-i]=w[i];
cout <<s;
return 0;
}
```

Ushbu kodni kompyuterda tekshirib ko‘rish o‘quvchilarning o‘ziga havola etinladi.

4-masala. Guruhdagi 10 talabaning informatika fani bo‘yicha imtihonda olgan baholari berilgan bo‘lsin. Har bir talabaning bahosi guruhning o‘rtacha o‘zlashtirish bahosidan qanchaga farq qiladi?

Yechish g‘oyasi. Har bir talaba olgan baholarni kiritiladi va ularning umumiy yig‘indisi hisoblanadi. So‘ngra umumiy yig‘indini talabalar soni 10 ga bo‘lib, o‘rtacha o‘zlashtirish aniqlanadi. Har bir talabaning bahosidan o‘rtacha o‘zlashtirish bahosini ayirib, oradagi farq topiladi.

```
# include <iostream.h>
int main()
{
int i; int baho[10]={56, 88, 75, 65, 78, 82, 79, 81, 68, 72};
float s=0;
for (i=0; i<=9; i++) s=s+baho[i];
s=s/10;
cout<<“o‘rtacha baho= “<<s<<”\n”;
for (i=0; i<=9; i++)
cout<<i<<“-talaba uchun o‘rtacha farq=”<<baho[i]−s << “\n”;
return 0;
```

}

Ushbu dasturni tekshirib ko‘rish uchun namuna tariqasida ballarni 1 bilan 100 orasida tanlanadi (kodning 4-satriga e’tibor bering). Bu ma’lumotlar uchun kod quyidagi natijani beradi:

O‘rtacha baho= 74.400002
0-talaba uchun o‘rtacha farq= −18.400002
1-talaba uchun o‘rtacha farq= 13.599998
2-talaba uchun o‘rtacha farq= 0.599998
3-talaba uchun o‘rtacha farq= −9.400002
4-talaba uchun o‘rtacha farq= 3.599998
5-talaba uchun o‘rtacha farq= 7.599998
6-talaba uchun o‘rtacha farq= 4.599998
7-talaba uchun o‘rtacha farq= 6.599998
8-talaba uchun o‘rtacha farq= −6.400002
9-talaba uchun o‘rtacha farq= −2.400002

Eslatma. 1. Massiv albatta e’lon qilinishi lozim, aks holda uning elementlari bilan ishlab bo‘lmaydi. 2. Massiv elementlarini *for* sikli yordamida kiritish qulay hisoblanadi. 3. Elementlarning indeksi belgilangan diapazondan chetga chiqmasligini nazorat qilish lozim.

6.2. Massiv elementlarini tartiblash

Massivlar haqidagi masalalar ichida eng ko‘p uchraydigani bu uning elementlarini o‘sish yoki kamayish tartibida tartiblash masalasidir. Bu muammoni hal qilishning usullari ko‘p bo‘lib, ulardan birortasini boshqasidan ustun qo‘yib bo‘lmaydi. Har bir usul elementlar joylashuviga ko‘ra boshqasidan yaxshi bo‘lishi mumkin.

1-usul. $A[1:N]$ massiv elementlarini o‘sish tartibida tartiblash talab qilingan bo‘lsin.

Berilgan massiv elementlari ichidan eng kichigini topib, uning o‘rni 1-element bilan almashtiriladi. Shu bilan 1-element tartiblandi. Endi 2-element tartiblanadi. Buning uchun qolgan elementlar ichidan eng kichigi topilib, uning o‘rni 2-element bilan almashtiriladi

va hokazo. Bu jarayon *N*-1 marta takrorlanganda tartiblash ham tugaydi. Bu usul uchun kod quyidagicha yoziladi:

```
# include <iostream.h>
int main()
{
int min, a, j, k;
int i; int b[10]={56, 88, 75, 65, 78, 82, 79, 81, 68, 72};
for (i=0; i<=8; i++) {
min=b[i]; k=i;
for (j=i+1; j<=9; j++) if (b[j]<min) {min=b[j]; k=j;}
a=b[i]; b[i]=b[k]; b[k]=a;}
for (i=0; i<=9; i++)
cout<<b[i]<<" , ";
return 0;
}
```

Bu usul *eng kichik elementni chiqarish usuli* deyiladi.

2-usul. ***Ko‘piksimon usul***. Bu usulning asosiy g‘oyasi yengil elementlarni yuzaga chiqarishdan iborat bo‘lib, xuddi suv ichidan chiqayotgan pufakchalarni eslatadi. Yengil elementlar “vazni” darajasida borgan sari yuqorilab boradi. Buning uchun birinchi elementdan boshlab, hamma elementlari o‘z yonida turgan element bilan taqqoslanadi. Agar $a[i] > a[i+1]$ bo‘lsa, u holda bu elementlar o‘rni o‘zaro almashtiriladi. Tekshirish yana boshidan boshlanadi.

```
# include <iostream.h>
int main()
{  int a, i, k;
int b[10]={56, 88, 75, 65, 78, 82, 79, 81, 68, 72};
for (i=0; i<=8; i++)
if (b[i]>b[i+1]) {a=b[i]; b[i]=b[i+1]; b[i+1]=a; i=-1;}
for (i=0; i<=9; i++)
cout<<b[i]<<  " ,";
```

return 0;
}

***5-masala*: *B*(5, 5) butun sonli massiv berilgan bo‘lsin. Uning har bir satridagi eng katta elementlar orasidagi eng kichigi topilsin.**

Yechish g‘oyasi. Dastlab har bir satrdagi eng katta elementlardan *A*[5] massivni hosil qilamiz. So‘ngra *A* ning eng kichik elementi topiladi. Bu g‘oyaga mos kod quyidagicha yoziladi:

```
# include <iostream.h>
int main()
{ int a[5];
int b[5][5]={2, 4, 4, 2, 3, 4, 5, 4, 3, 2, 6, 8, 9, 1, 2, 3, 4, 5, 4, 3,
4, 4, 4, 8, 7};
for (i=0; i<=4; i++) {
max=b[i][0];
for (j=1; j<=4; j++)
if (max<b[i][j]) max=b[i][j] ; a[i]=max;}
kichik=a[0];
for (i=1; i<=4; i++) if (kichik>a[i]) kichik=a[i];
cout<< “Izlangan element=”<<kichik<<“ ”;
return 0;
}
```

Ushbu kod uchun kompyuter quyidagi natijani beradi:

Izlangan element= 4

6.3. Massivlardan satrlarni qayta ishlashda foydalanish

Biz yuqorida massivlardan satrlarni ham qayta ishlash uchun foydalanish mumkinligi haqida fikr bildirgan edik. Buning uchun bizga belgili tipdagi massivlar qo‘l keladi.

Belgili massivlarni ikki xil usulda e’lon qilish mumkin:

a) *char satr[] = “first”;*

b) *char satr[5]={’f’, ’i’, ‘r’, ‘s’, ‘t’, ’\0’};*

C++ tilida *a*-usul bilan e’lon qilingan massiv elementlarining soni ko‘rsatilmagan bo‘lsa ham qiymatidagi belgilar sonidan kelib

chiqqan holda *satr* massivini 6 elementli deb qabul qilinadi. Oxirgi elementni C++ tili to‘g‘ridan-to‘g‘ri nol element (\0) sifatida tan oladi va u satrning tugaganligini anglatadi. Satrlarni ifodalovchi barcha belgili massivlar ana shu belgi bilan tugashi shart.

Satr (yoki matn) belgili massiv bo‘lgani uchun uning alohida elementlariga sonli massivlardagi kabi indekslardan foydalangan holda murojaat qilish mumkin. Masalan, yuqoridagi misollarda *satr[0]* element “*f*” harfiga, *satr[6]* esa nol elementga (“\0”) mos keladi.

Shuningdek, satrli ma’lumotlarni ***cin*** operatori yordamida klaviatura orqali ham kiritish mumkin. Masalan,

char satr[20];

buyrug‘i uzunligi 20 ta, eng yuqori indeksi 19 bo‘lgan belgili massivni e’lon qiladi. Unga klaviatura orqali qiymat berish uchun

cin>>satr;

buyrug‘idan foydalanish mumkin. Kompyuter bu buyruqqa javoban klaviaturadan kiritilgan satrning oxiriga “\0” belgisini qo‘shib o‘qiydi. Shuni yoddan chiqarmaslik kerakki, klaviaturadan kiritilgan bu satrning uzunligi 20 dan kichik bo‘lishi mumkin, aks holda, dasturni bajarilish vaqtidagi xatolik yuzaga keladi.

Belgili massiv elementlarini ***cout*** yordamida ekranga chiqarish ham mumkin. Buning uchun

cout << satr;

ko‘rsatmasi yetarli. Bu holda indekslar diapazonini ko‘rsatish shart emas. Faqat zarur bo‘lganda bo‘sh joy (probel) belgilari o‘rniga “_” belgisini qo‘yish lozim bo‘ladi. Quyidagi dastur kodiga e’tibor bering.

```
# include <iostream.h>
int main()
{
char satr1[20];
char satr2[]=”Assalomu alaykum”;
cout <<“Uzunligi 20 tagacha bo‘lgan matn kiriting\n”;
cin >>satr1;
cout<<satr2<<“\n”;
```

```
for (int i=0; satr1[i]!= '\0'; i++)
cout<<satr1[i];
cout<<"\n";
return 0;
}
```

Ushbu dasturni ishga tushiramiz:

```
Uzunligi 20 tagacha bo‘lgan matn kiriting
C++_ga_xush_kelibsiz
Assalomu alaykum
C++_ga_xush_kelibsiz
```

Tekshirish uchun savol va topshiriqlar

1. Massiv nima?
2. Qanday turdagi massivlarni bilasiz?
3. C++ tilida massivlar qanday e’lon qilinadi?
4. Massivning biror elementiga qanday murojaat qilish mumkin?
5. Quyidagi masalalar uchun dastur ishlab chiqing.

a) Natural N soni va $A(1{:}N)$ butun sonlar jadvali berilgan. Uning nomeri toq, o‘zi juft bo‘lgan elementlar yig‘indisi topilsin.

b) $A(1{:}30)$ haqiqiy sonlar jadvali berilgan. Hisoblang:

$$max(a_1+a_{30}, a_2+a_{29}, ..., a_{15}+a_{16}).$$

c) Natural N soni va $A(1{:}N)$ butun sonlar jadvali berilgan. Unda necha xil element uchraydi? (Bir xil elementlar bitta element deb hisoblanadi.)

d) $N\times M$ o‘lchovli A matritsaning har bir satridagi eng katta elementlar ro‘yxatini aniqlang.

e) $N\times M$ o‘lchovli A matritsaning har bir satridagi eng kichik elementlar orasidan eng kattasi topilsin.

7-§. FUNKSIYALAR BILAN ISHLASH

7.1. Formal, joriy va lokal o‘zgaruvchilar

Ko‘pincha masalalar uchun dastur ishlab chiqish jarayonida

masala qanday o‘zgaruvchilar uchun berilganligi ko‘rsatilmaydi. Lekin uni yechish uchun shartli ravishda o‘zgaruvchilar tanlab olinadi va ana shu o‘zgaruvchilar uchun qo‘yilgan masalani to‘la yechish qonun-qoidalari (algoritmi) ishlab chiqiladi. Bunday o‘zgaruvchilar formal o‘zgaruvchilar deyiladi. Masalan, kvadrat tenglama uchun *a, b* va *c* koeffitsiyentlar formal o‘zgaruvchilar deb qaraladi.

Shu sinfga taalluqli bo‘lgan konkret masalada esa odatda barcha o‘zgaruvchilar yoki ularning qiymatlari aniq ko‘rsatib qo‘yiladi va masalani ana shu qiymatlar uchun hal qilish talab qilinadi. Bunday o‘zgaruvchilar joriy o‘zgaruvchilar deb ataladi. Endi masalani yechish uchun yaratilgan hamma qonun-qoidalarni formal o‘zgaruvchilar o‘rniga masalada berilgan joriy o‘zgaruvchilarga qo‘llash lozim bo‘ladi.

Masala. Bo‘yi *N*, eni *M* bo‘lgan to‘g‘ri to‘rtburchak yuzi topilsin.

Yechish g‘oyasi. Ma’lumki, to‘g‘ri to‘rtburchak yuzi $S=A\cdot B$ formula bilan topiladi. Bu yerda *A* – to‘rtburchak bo‘yi, *B* – eni. Ammo bizga berilgan masalaning shartiga ko‘ra, $S=N\cdot M$. Bu misolda *A* va *B* formal o‘zgaruvchi, *N* va *M* esa joriy o‘zgaruvchi hisoblanadi. Formal o‘zgaruvchi bilan joriy o‘zgaruvchi ustma-ust tushishi ham mumkin.

Masala shartida ko‘rsatilmagan, lekin masalani yechish uchun hisoblanishi zarur bo‘lgan o‘zgaruvchilar oraliq o‘zgaruvchilar deyiladi.

Oraliq o‘zgaruvchilar faqat bitta protseduraga taalluqli bo‘ladi. Shuning uchun ularni lokal (mahalliy) o‘zgaruvchilar deb ham yuritiladi. Masalan, kvadrat tenglamani yechishda diskriminantni ifodalovchi *D* o‘zgaruvchi lokal hisoblanadi.

7.2. Funksiyalarni e’lon qilish va foydalanish

C++ tili boshqa dasturlash tillaridan faqat funksiyalar bilan ishlay olishi bilan farqlanadi.

C++ tilida funksiya eng asosiy tushunchalardan biri sanaladi. Chunki, birinchidan, ixtiyoriy dastur hech bo‘lmaganda *main* nomli

bitta funksiyani (asosiy funksiya) o‘z ichiga oladi. Aynan shu funksiya dasturga kirish nuqtasini belgilab beradi. Dastur kodiga *main* funksiyasidan tashqari bir qator yordamchi funksiyalar ham kirishi mumkin. Ularning barchasi *main* funksiyasi yordamida boshqariladi. Bu funksiyalarning hammasi global hisoblanadi.

Ayrim ifodalarning qiymatlarini hisoblashda qandaydir funksiyalardan foydalanishga to‘g‘ri keladi. Masalan, $y = \cos x + 3$ ifodaning qiymatini hisoblaganda, avval $\cos x$ o‘rniga uning qiymati qo‘yiladi va 3 ga qo‘shiladi. Agar $\cos x$ funksiyaning o‘rnida murakkab funksiya yoki uzundan-uzun arifmetik ifoda tursachi?

Bunday hollarda dasturchi ishini soddalashtirish uchun C++ tilida funksiya yoki foydalanuvchi funksiyasini qurish imkoniyati yaratilgan. Funksiyalar formal o‘zgaruvchilar uchun tashkil qilinadi.

Funksiyalar odatda murakkab ifodalar, uzun arifmetik ifodalar yoki biror ifoda qiymatlarini argumentlarining turli qiymatlari uchun qayta-qayta hisoblashga to‘g‘ri kelgan hollarda tashkil qilish dasturchi ishini osonlashtiradi.

Yana shunday masalalar ham mavjudki, ularni yechish jarayonida bitta masalani bir nechta kichik masalalarga ajratish qulay hisoblanadi. Agar hosil bo‘lgan masalalar bitta sinfga tegishli bo‘lsa yana ham yaxshi. Bu holda bir sinfga tegishli bo‘lgan har bir masalaga alohida dastur yozish o‘rniga, ulardan bittasi uchun formal o‘zgaruvchilar o‘ylab topiladi va masalani ana shu o‘zgaruvchilar ychun yechish buyruqlari ketma-ketligi ishlab chiqiladi. Funksiyalarni bunday masalalarga nisbatan qo‘llash ham mumkin. Bu holda funksiya qiymati kichik masalaning yechimiga teng bo‘ladi.

Bitta dastur tarkibidagi barcha funksiyalar ishini boshqaradigan dastur asosiy deb ataladi.

Agar dasturda funksiyalardan foydalanish rejalashtirilgan bo‘lsa, har bir funksiya uchun asosiy dasturdan o‘tadigan ma’lumotlar (o‘zgaruvchilar) ro‘yxati hamda qiymati funksiyadan asosiy dasturga qaytishi lozim bo‘lgan o‘zgaruvchi aniqlab olinadi.

Funksiyaga undagi formal o‘zgaruvchilar o‘rniga joriy o‘zgaruvchilarni ko‘rsatib (ro‘yxatdagi 1-formal o‘zgaruvchi o‘rniga 1-joriy

o‘zgaruvchi, 2-formal o‘zgaruvchi o‘rniga 2-joriy o‘zgaruvchi va hokazo) murojaat qilish mumkin.

Funksiyaga murojaat qilinganda, asosiy dasturning bajarish jarayoni to‘xtaydi va kompyuter funksiyani bajara boshlaydi. Funksiyadagi barcha buyruqlar joriy o‘zgaruvchilar uchun to‘la bajarilgandan so‘ng, kompyuter yana asosiy dasturning kelgan qismidan boshlab navbatda turgan buyruqlarni bajarishda davom etadi.

Funksiyalar bilan ishlash imkoniyatiga ega bo‘lish uchun ularga birinchi marta murojaat qilishdan avval aniqlangan bo‘lishi lozim. Funksiyani aniqlashda unga murojaat qilganda bajarilishi lozim bo‘lgan amallar ketma-ketligi, foydalanilgan o‘zgaruvchilarning tiplari, funksiya nomi hamda asosiy dasturga qaytarilishi lozim bo’lgan ma’lumot (natija) tipi ko‘rsatiladi. Bunda funksiya nomi va rasmiy o‘zgaruvchilar ro‘yxati funksiyaning umumiy belgisi hisoblanadi va ana shu belgiga ko‘ra funksiyaga murojaat qilinadi.

Funksiyani aniqlash uning umumiy belgilari va jismini ko‘rsatishdan iborat bo‘lib, quyidagicha tuzilmaga ega:

funksiya_tipi funksiya_nomi (ro‘yxat);
{
funksiya_jismi
}

Bu yerda *funksiya_tipi* – funksiya qaytaradigan ma’lumotning tipi (agar funksiya ma’lumot qaytarmasa bu tip *void* bo‘ladi); *funksiya_nomi* o‘zgaruvchi-identifikatorni anglatadi. Bu nom boshqa o‘zgaruvchilar kabi takrorlanmas bo‘lishi lozim; *ro‘yxat* yoki bo‘sh, yoki *void*, yoki alohida ko‘rsatiladigan rasmiy o‘zgaruvchilarning tiplari va nomlarini o‘z ichiga olishi mumkin; *funksiya_jismi* turli amallar va ko‘rsatmalar ketma-ketligidan iborat bo‘lib, odatda yuqorida ta’kidlanganidek, alohida olingan kichik bir masalani hal qilishga qaratiladi. Jismning so‘nggi bajariladigan buyrug‘i *return* bo‘lib, u boshqaruvni funksiyaga murojaat qilish nuqtasiga uzatish (qaytarish) amalini bajaradi. Bu buyruq umumiy ko‘rinishda

return ifoda;

yoki

return;

ko‘rinishida yoziladi. *Return* operatoridan keyin ko‘rsatilgan ifoda funksiyadan asosiy dasturga uzatiladigan qiymatni belgilab beradi. Bu operator funksiyadan hech qanday qiymatni qaytarish ko‘zda tutilmagan hollarda (ya’ni *void* tipida bo‘lsa) yozilmaydi.

Bitta funksiya jismida bir nechta *return* buyruqlaridan foydalanish mumkin. C++ tilida agar dastur o‘z ishini muvaffaqiyatli yakunlasa, 0 qiymatini qaytaradi.

Funksiyalarni e’lon qilishga namunalar keltiramiz:

```
void print (char x, int y)      // hech narsa qaytarilmaydi
{
cout “”\n” “ x “ y;    // return tushirib qoldirilgan
}
float min(float a, float b);   // funksiya natijasi float tipida
 { if a<b) return a;      // a va b sonlaridan kichigini qaytaradi }
```

Funksiyaga murojaat qilganda rasmiy o‘zgaruvchilar joriy o‘zgaruvchilar bilan almashtiriladi va bunda tiplarining o‘zaro mosligi qat’iy nazorat qilinadi. Bunday moslik bo‘lmaganda C++ tilida o‘zgaruvchilar tiplarini to‘g‘ridan-to‘g‘ri almashtirish ham ko‘zda tutilgan. C++ haqida fikr yuritganda, “tiplarning qat’iy mosligi”ga alohida e’tibor beriladi. Shunga ko‘ra, rasmiy va joriy o‘zgaruvchilar tiplarining o‘zaro mosligi kompilyatsiya qilish jarayonidayoq tekshiriladi.

Funksiyaga murojaat qilish (sodda qilib aytganda, uni chaqirish) oddiy qavslar yordamida amalga oshiriladi. Qavslar ichida esa joriy o‘zgaruvchilar ro‘yxati ko‘rsatiladi:

funksiya_nomi (joriy o‘zgaruvchilar ro‘yxati);

Funksiyaga murojaat qilganda tipi funksiya tipi bilan bir xil bo‘lgan qiymatga ega bo‘ladi.

Joriy o‘zgaruvchilar (funksiya argumentlari) va rasmiy o‘zgaruvchilar o‘rtasidagi moslik rasmiy va joriy o‘zgaruvchilarning ro‘y-

xatdagi o‘rniga ko‘ra aniqlanadi.

Joriy o‘zgaruvchilar murojaat qiluvchi dastur tomonidan uzatiladi va funksiya jismidagi ko‘rsatmalarni bajarishda ana shu o‘zgaruvchilarning qiymatlaridan foydalaniladi.

Shunday qilib, joriy o‘zgaruvchilar ro‘yxati yoki bo‘sh, yoki *void*, yoki vergul bilan ajratilgan o‘zgaruvchilar ro‘yxatidan iborat bo‘lishi mumkin.

Yuqoridagi fikrlarni amaliyotga tatbiq etishga urinib ko‘raylik.

1-masala. Haqiqiy *a*, *b*, *c* va *d* sonlari berilgan bo‘lsin. Ularning eng kichigini toping.

Yechish g‘oyasi. Dastlab, *a* va *b* sonlarining eng kichigini aniqlaymiz va uni *p* bilan belgilaymiz. So‘ngra, *c* va *d* larning eng kichigini *q* bilan belgilaymiz. Ishning yakunida *p* va *q* larning eng kichigi topiladi. Ko‘rinib turibdiki, ikki sonning eng kichigini topish masalasidan 3 marta foydalanilmoqda. Shuning uchun rasmiy o‘zgaruvchi sifatida *n* va *m* sonlarni tanlab olib, ularning kichigini topish uchun funksiya tashkil qilinadi. Bu mulohazalarni e’tiborga olib, kodni quyidagicha yoziladi:

```
# include <iostream.h>
float min(float n, float m)
{
float k;
if (n<m) k=n; else k=m;
return k;
}
void main()
{ float a, b, c, d;
cin >>a>>b>>c>>d;
float p, q, kichik;
p=min(a, b);
q=min(c, d);
kichik=min(p, q);
```

```
cout<<kichik;
return;
}
```

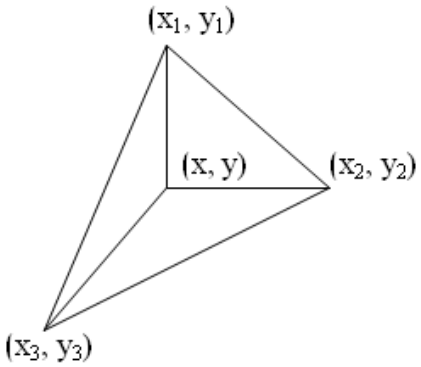
Eslatma. *Return* buyrug‘idan bir necha marta foydalanish mumkin bo‘lgani uchun funksiyani

```
float min(float n, float m)
{ if (n<m) return n; else return m;  }
```

ko‘rinishida ham yozish mumkin.

C++ tilida bitta funksiya tarkibida boshqa funksiyalarga ham murojaat qilish mumkin.

2-masala. Uchburchak uchlarining koordinatalari (*x1*,*y1*), (*x2*,*y2*), (*x3*,*y3*) berilgan bo‘lsin. (*x, y*) koordinatali nuqta shu uchburchakka tegishli bo‘la oladimi?



Yechish g‘oyasi. Berilgan uchburchak yuzasi *S* bo‘lsin. Uchburchak uchlarini (*x, y*) koordinatali nuqta bilan tutashtirib, 3 ta uchburchak hosil qilamiz. Ularning yuzalari mos ravishda *S1, S2* va *S3* bo‘lsin. Agar (*x, y*) nuqta berilgan uchburchak ichida yotsa, *S=S1+S2+S3* bo‘ladi. Aks holda nuqta uchburchak ichida yotmaydi.

Ko‘rinib turibdiki, bu yerda uchlarining koordinatalari ma’lum bo‘lgan to‘rtta uchburchak yuzini hisoblashga to‘g‘ri kelmoqda. Bu koordinatalar uchun formal o‘zgaruvchilarni (*a1, b1*), (*a2, b2*) va (*a3, b3*) tarzida tanlash mumkin. Uchburchakning tomonlari va yarim perimetrini belgilash uchun mos ravishda *A, B, S* va *P* o‘zgaruvchilar olinadi. Ular masala shartida ko‘rsatilmagani uchun lokal o‘zgaruvchilar hisoblanadi.

Tanlangan formal o‘zgaruvchilarni hisobga olib, uchburchak yuzini topish buyruqlaridan iborat funksiya hosil qilinadi. Funksiyada uchlarining koordinatalari ma’lum bo‘lgan kesma uzunligi uch marta hisoblanishi lozim bo‘lgani uchun kesma uzunligini hisoblash

maqsadida alohida funksiya tashkil qilish mumkin. Shundan keyin

(*x1, x2, x3, y1, y2, y3*), (*x, x1, x2, y, y1, y2*),

(*x, x2, x3, y, y2, y3*) , (*x, x1, x3, y, y1, y3*)

joriy o‘zgaruvchilar uchun yuzani hisoblash funksiyasiga murojaat qilib, uchlari ana shu nuqtalarda yotgan uchburchaklarning *S1, S2, S3* va *S* yuzalari hisoblanadi. Yuzani hisoblash funksiyasi har gal bajarilganda kesma uzunligini hisoblash funksiyasiga uch marta murojaat qiladi. Uchburchaklarning yuzalari topilganidan so‘ng, *S=S1+S2+S3* munosabatning rost yoki yolg‘on bo‘lishiga qarab xulosa chiqariladi. Mazkur jarayon uchun kod quyidagicha yoziladi:

```
# include <iostream.h>
# include <math.h>
float kesma(float a1, float b1, float a2, float b2)
{
float k=sqrt((a2-a1)*(a2-a1)+(b2-b1)*(b2+b1));
return k;
}
float yuza(float a1, float b1, float a2, float b2, float a3, float b3)
{
float n1=kesma(a1, b1, a2, b2);
float n2=kesma(a1, b1, a3, b3);
float n3=kesma(a2, b2, a3, b3);
float p=(n1+n2+n3)/2;
float yuza=sqrt(p*(p-n1)*(p-n2)*(p-n3));
return yuza;
}
void main()
{
float x, x1, x2, x3, y, y1, y2, y3;
cin >>x>>y;
cin >>x1>>x2>>x3>>y1>>y2>>y3;
float s1=yuza(x, y, x1, y1, x2, y2);
```

```
float s2=yuza(x, y, x1, y1, x3, y3);
float s3=yuza(x, y, x2, y2, x3, y3);
float s=yuza(x1, y1, x2, y2, x3, y3);
if (s==s1+s2+s3) cout <<”Ha”; else cout <<”Yo‘q”;
return;
}
```

7.3. Funksiyalarni qayta yuklash

Ma’lumki, funksiyalarni aniqlashda ularning qaytarishi lozim bo‘lgan qiymatlar tipi va funksiya uchun zarur bo‘lgan parametrlar tipini ko‘rsatish lozim edi.

Faraz qilaylik, ikkita butun sonni qo‘shish uchun funksiya qurilgan bo‘lsin. Agar uchta butun sonni qo‘shish talab qilingan bo‘lsa, ular uchun boshqa nomdagi funksiyani qurish talab qilinadi. Ikkita haqiqiy sonni qo‘shish uchun esa boshqa funksiya qurish lozim bo‘ladi.

Bunday hollarda bir xil funksiyani takror va takror yozishning o‘rniga, C++ tili bir xil nomdagi funksiyalarni qurish imkonini beradi. Dasturni kompilatsiya qilish jarayonida C++ funksiyalarning har biridagi argumentlar miqdori e’tiborga olinadi va aynan kerak bo‘lgan funksiyani chaqiradi. Kompilyatorga bir nechta funksiyalar orasidan keragini tanlash imkoniyati funksiyalarni qayta yuklash deb ataladi.

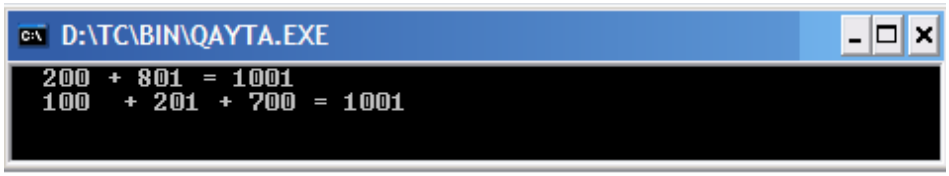
Funksiyalarni qayta yuklash amali bir xil nomdagi parametrlarni har xil tipga mansub bo‘lgan turli funksiyalar uchun qo‘llashga ruxsat beradi.

Masalan, quyidagi dastur *add_values* nomli ikkita funksiyani qayta yuklash uchun xizmat qiladi:

```
#include <iostream.h>
int add_values (int a, int b)
{
  return(a + b);
}
int add_values (int a,  int b, int c)
```

```
{
  return(a + b + c);
}
void main(void)
{
 cout << "200 + 801 = " << add_values(200, 801) << endl;
 cout << "100 + 201 + 700 = " << add_values(100, 201, 700)
  << endl;
}
```

Ushbu dastur quyidagi natijani beradi:



Ko‘rinib turibdiki, dasturda ikkita bir xil nomdagi, ammo parametrlari soni har xil bo‘lgan *add_values* funksiyasi aniqlangan. Bu holda kompilyator parametrlar soniga ko‘ra qaysi funksiyani qo‘llash haqida mustaqil ravishda xulosa qiladi.

Quyidagi misolga e’tibor bering. Unda *show_message* funksiyasi qayta yuklanadi. Birinchi *show_message* funksiyasiga parametrlar uzatilmaydi va u ekranga standart axborotni chiqaradi. Ikkinchisi unga uzatilgan bitta ma’lumotni, uchinchisi esa ikkita ma’lumotni ekranga chiqaradi.

```
#include <iostream.h>
 void show_message(void)
 {
 cout << " Standart axborot: " << " C++ da dastrulashni
o‘rganamiz" << endl;
 }
 void show_message(char *message)
 {
   cout << message << endl;
 }
 void show_message(char *first, char *second)
```

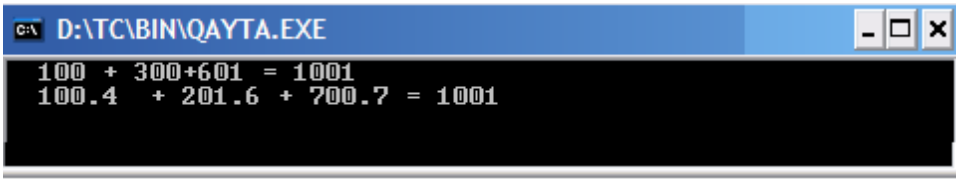
```
{  cout << first << endl;
   cout << second << endl;  }
void main(void)
{
   show_message();
   show_message(“ C++ tili da dastrulash oson”);
   show_message(“ C++ tili zo‘r til!”,“Qayta yukalsh yaxshi imkoniyat!”) ;
}
```

Bu dastur quyidagi natijani beradi:

Qayta yuklanadigan funksiyalar bir xil tipdagi qiymatlarni qaytarishi lozim, ammo parametrlarining miqdori va tiplari har xil bo‘lishi ham mumkin. 1-misolga kichik o‘zgarish kiritamiz:

```
#include <iostream.h>
int add_values (int a,  int b, int c)
{
    return(a + b + c);
}
void main(void)
{
   cout << “  100 + 300+601 = “ <<
   add_values(100, 300, 601) << endl;
   cout << “  100.4  + 201.6 + 700.7 = “
   << add_values(100.4, 201.6, 700.1) << endl;
 }
```

Dasturning natijasi quyidagicha bo‘ladi:



Takrorlash uchun savol va topshiriqlar

1. Formal va joriy o‘zgaruvchilarning farqini tushuntiring.
2. Oraliq o‘zgaruvchi nima va undan qachon foydalaniladi?
3. Funksiyalar qachon va qanday tashkil qilinadi?
4. Funksiyalarning umumiy tarkibini aytib bering.
5. Funksiyalardan qanday foydalanish mumkin?
6. Funksiyalarni qayta yuklash nima?
7. Quyidagi masalalar uchun dastur ishlab chiqing.

a) *k*, *l* va *m* natural sonlari hamda $x_1, \ldots, x_n, y_1, \ldots, y_m, z_1, \ldots, z_m$ haqiqiy sonlari berilgan bo‘lsin. Hisoblang:

$$t = \begin{cases} (\max(x_1,\ldots,x_k) + \max(z_1,\ldots,z_m))/2, & agar\ \max(x_1,\ldots,x_k) \ge 0 \\ \min(y_1,\ldots,y_{l)} + \max(z_1,\ldots,z_m), & boshqa\ \ hollarda \end{cases}.$$

b) *s* va *t* haqiqiy sonlari berilgan bo‘lsin. Hisoblang:

$$h(s,t)+max(h^2(s-t,\ st),\ h^4(s-t,\ s+t))+h(1+s,\ 1+t).$$

Bu yerda $h(a,b) = \dfrac{a}{1+b^2} + \dfrac{b}{1+a^2} - \dfrac{a+b}{ab} + 2$.

c) *a*, *b*, *c* va *d* natural sonlari berilgan. Bu sonlar uchun *a/b* va *c/d* kasrlarni qisqarmaydigan ko‘rinishga keltiring. (Ikki natural sonning eng katta umumiy bo‘luvchisini topish funksiyasidan foydalaning.)

d) $x_1, y_1, \ldots, x_{10}, y_{10}$ haqiqiy sonlari berilgan bo‘lsin. O‘nburchak uchlarining koordinatalari mos ravishda $(x_1,y_1), \ldots, (x_{10},y_{10})$ bo‘lsin. Shu o‘nburchakning perimetrini toping. (Koordinatalari berilgan kesma uzunligini topish funksiyasidan foydalaning.)

e) $N > 2$ natural soni berilgan bo‘lsin. Bu son uchun Goldbax gipotezasini tekshiring. (Sonni tub yoki tub emasligini topish funksiyasidan foydalaning.)

8-§. REKURSIYA

8.1. Rekursiya tushunchasi

Agar funksiya o‘zidan yordamchi funksiya sifatida foydalanadigan bo‘lsa, bunday funksiyalar rekursiv deyiladi.

Rekursiv funksiyalar ikki turga bo‘linadi:

a) to‘g‘ri rekursiya. Bunda dastur o‘ziga-o‘zi murojaat qiladi.

b) yondosh rekursiya. Bunda *A* funksiya *B* ga, *B* funksiya *A* ga murojaat qiladi.

Rekursiv funksiya yozish uchun avvalo: 1) rekkurent munosabat; 2) shu munosabat uchun boshlang‘ich holatlar aniqlangan bo‘lishi shart.

Rekkurent munosabat deganda qaralayotgan jarayonga doir muayyan bosqichlarni avvalgi bosqichlar bilan bog‘lovchi munosabatlar tushuniladi. Masalan, $N!=N\cdot(N-1)!$ formulani $N!$ uchun rekkurent munosabat deb qarash mumkin. Boshlang‘ich holat sifatida esa 1!=1 olinadi.

Keltirilgan ma’lumotlarni hisobga olsak, faktorialni hisoblash masalasi uchun rekkurent va boshlang‘ich munosabatlar quyidagicha bo‘ladi:

$$N! = \begin{cases} N\cdot(N-1)!, & agar\ N>1 \\ 1, & agar\ N=1 \end{cases}$$

Ko‘rinib turibdiki, $N!$ ni hisoblash uchun $(N-1)!$ ma’lum bo‘lishi kerak. Lekin, $(N-1)!=(N-2)!\cdot(N-1)$ bo‘lgani uchun o‘z navbatida $(N-2)!$ ni topish talab qilinadi. $(N-2)!$ esa $(N-3)!\cdot(N-2)$ ga teng va hokazo. Bu yerda $N!$ ni hisoblash algoritmi o‘zining ichiga o‘zi “cho‘kib” borishi hodisasi ro‘y bermoqda. Cho‘kish jarayoni boshlang‘ich holat sodir bo‘lgunga qadar, ya’ni 1! gacha davom etadi. Shundan keyin, “cho‘kish” jarayoni to‘xtaydi, 1!=1 ekanligi haqida ko‘rsatma olgan kompyuter yuqoriga qarab “suzib” chiqish bosqichini boshlaydi. Ya’ni, $2!=1$, $2!=1\cdot2=2$, $3!=2!\cdot3=6$ va hokazo. Bu holat to $N!$ hisoblanmaguncha davom etaveradi.

Yuqorida keltirilgan jarayon dasturi quyidagicha yoziladi:

```
# include <iostream.h>
 long fak(int m)
 { long f;
 if (m==1) f=1; else f=fak(m-1)*m;
 return f;
 }
 void main()
 {
 int n;
 cout << "Butun sonni kiriting";
 cin>>n;
 cout<<fak(n);
 return;
 }
```

N=4 uchun "cho'kish" va "suzib chiqish" jarayoni quyidagicha:

N ning qiymatlari	4 4 3 2 1	Rekursiyadan chiqishdagi oraliq qiymatlar
N=1 sharti natijasi	Yo'q yo'q yo'q ha	
	fak(3)*4 *fak*(2)*3 *fak*(1)*2 1	*p:*= *p**4=24 *p:*= *p**3=6 *p:*= *p**2=2 *p:*= 1

Demak, EHM *N*=4 bo'lgan hol uchun 24 natijani beradi.

Shunday masalalar sinfi mavjudki, ularni rekursiyadan foydalanmay turib yechishning boshqa usullari yo'q.

Masala. $f(n)$ funksiyaning qiymatlari $f(0)=1$, $f(2n)=f(n)$ va $f(2n+1)=f(n)+1$ ifodalar yordamida topiladi. Berilgan k natural soni uchun *f(k)* ni toping.

Yechish g'oyasi. Yuqoridagi masalani k ta elementli massiv

yordamida yechish ko‘pchilikning nazarida oson usulga o‘xshaydi. Lekin bu to‘g‘ri emas. Chunki, *k* yetarlicha katta son bo‘lsa, *k* ta elementli massiv kompyuter xotirasiga sig‘may qolishi mumkin. Qolaversa, siqqan taqdirda ham, bu elementlarning hammasidan foydalanilmaydi. Masalan, *k*=1000 bo‘lganda bu masalani yechish uchun massivning 1000 ta elementidan ko‘pi bilan 11 tasi kerak bo‘ladi (nima uchunligini o‘ylab ko‘ring), qolganlari esa kompyuter xotirasini befoyda band qiladi. Bunda xotiradan no’orin foydalanish holati yuz beradi va u keyinchalik salbiy oqibatlarga olib kelishi mumkin. Shuning uchun qo‘yilgan masalani massivdan foydalanmay yechish eng yaxshi usul hisoblanadi. Bu usulning mohiyati rekursiya mexanizmini qo’llashdan iborat.

Zarur bo‘lgan rekkurent munosabat va boshlang‘ich holatlarning masala shartida keltirilganligi ishni yanada osonlashtiradi.

```
# include <iostream.h>
int fun(int m)
{
int f;
if (m==0) f=1; else
{
int h=m/2;
if (m % 2==0) f=fun(h); else f=fun(h)+1;
}
return f;
}
void main()
{
int n;
cout <<”Butun sonni kiriting”;
cin >>n;
cout <<”f(“<<n<<”)=”<<fun(n);
return;
}
```

Bu dasturni *k*=11 bo‘lgan hol uchun bajargan EHM

$$f(11)=4$$

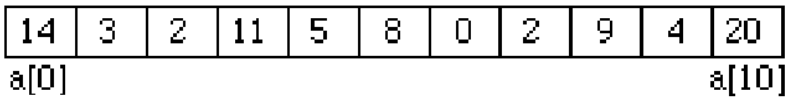
natijani ekranga chiqaradi.

Yuqoridagi ma’lumotlardan ko‘rinib turibdiki, rekursiya oddiy protsedura yoki funksiyaga nisbatan murakkabroq tushuncha, lekin u mohir dasturchi qo‘lida juda ham yaxshi vositaga aylanishi mumkin.

8.2. Tez saralashning rekursiv algoritmi

Rekursiyani amalda qo‘llash uchun namuna sifatida massiv elementlarini tez saralash algoritmi *QuickSort* ni ko‘rib chiqamiz. Ushbu algoritm massiv elementlarini o‘sish (yoki kamayish) tartibida tartiblash uchun xizmat qiladi.

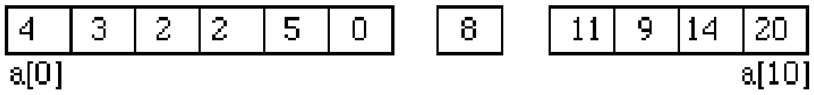
Faraz qilaylik, massivning 11 elementi quyidagicha joylashgan bo‘lsin (2-rasm).



2-rasm. Massivning boshlang‘ich holati.

Algoritm g‘oyasi massivni rekursiv asosda ikkiga ajratish va tartiblash amalini ularning har biri uchun bajarishni o‘z ichiga oladi. Ajratish chegaraviy deb ataladigan biror elementni tanlash orqali amalga oshiriladi. Massiv ikkiga ajratilganidan so‘ng, ular shunday qayta ishlanadiki, chegaraviy elementning bir tomonida undan kichik yoki teng bo‘lgan elementlar, ikkinchi tomonida esa katta yoki teng bo‘lgan elementlar joylashadi.

Agar chegaraviy element sifatida 8 ni tanlasak, birinchi tartib-lashdan keyin massiv 3-rasmdagi holga keladi. Endi xuddi shu jarayonni massivning o‘ng va chap qismlari uchun qo‘llanadi.



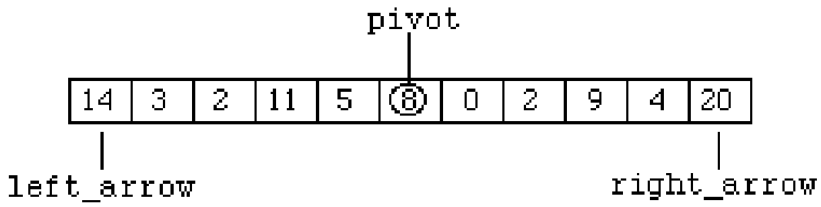
3-rasm. Massivning dastlabki saralashdan keyingi holati.

Massivni qismlarga ajratish va ularda tartiblash amalini rekursiv mexanizm yordamida bajarish mumkin. Buning uchun chegaraviy

element indeksi o‘rtadagi element indeksi shaklida hisoblanadi. Bunda “first” va “last” lar massiv qismining birinchi va oxirgi element-larining indekslarini angalatadi.

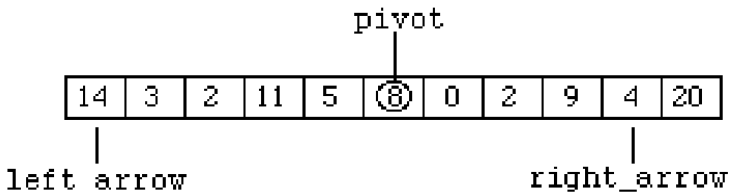
Massiv elementlarini rekursiv tartiblash protsedurasini batafsil tahlil qilamiz.

Massivning qayta ishlanayotgan qismi chetki elementlari indeks-larini “left_arrow” hamda “right_arrow” orqali belgilaymiz (4-rasm).



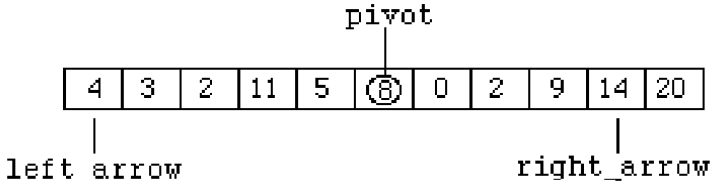
4-rasm. “left_arrow” va “right_arrow” larning tartiblashdan avvalgi indekslari.

Tartiblash jarayonida “left_arrow” va “right_arrow” indekslar chegaraviy elementlar tomonga qarab surilib boradi. Masalan, “right_arrow” indeksli element chegaraviy elementdan kichik yoki teng bo‘lib qolmaguncha chapga suriladi. Xuddi shuningdek, “left_arrow” indeksli element chegaraviy elementdan katta yoki teng bo‘lib qolmaguncha o‘ngga surilib boradi. Biz qarayotgan holda bu element massivning boshida joylashgan (5-rasm).



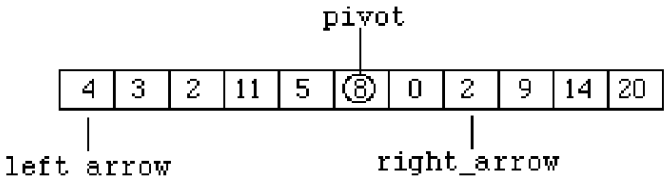
5-rasm. Almashtirish uchun chap va o‘ng elementlarni izlash.

Shundan keyin bu elementlar o‘zaro o‘rin almashadi (6-rasm).



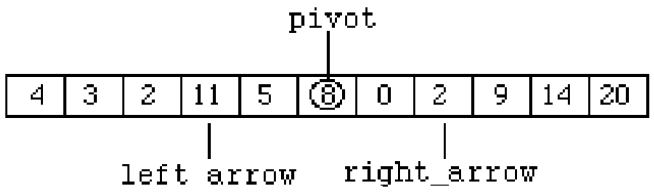
6-rasm. Ikki element o‘zaro o‘rin almashganidan keyingi holat.

Almashuvdan so‘ng, “right_arrow” chap tomonga qarab, to chegaraviy elementdan kichik yoki teng bo‘lgan element topilguncha surilib boradi. (7-rasm.)



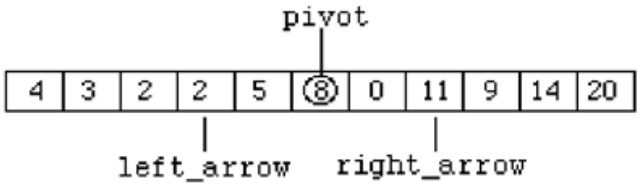
7-rasm. O‘ng tomonda o‘rin almashish uchun element topildi.

“left_arrow” indeks o‘rin almashtirish uchun zarur bo‘lgan element topilguncha o‘ng tomonga qarab surilib boradi (8-rasm).



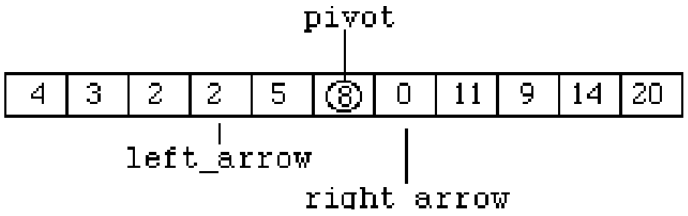
8-rasm. Chap tomonda almashish uchun element topildi.

So‘ngra topilgan elementlarning o‘rinlari o‘zaro almashtiriladi (9-rasm).



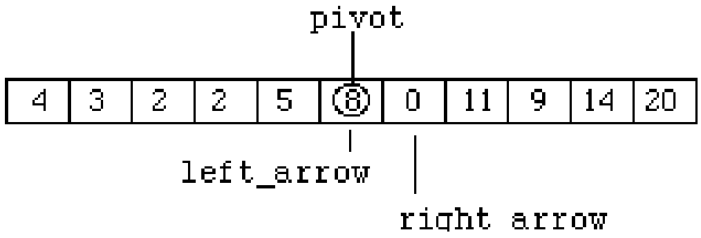
9-rasm. Ikkinchi marta o‘rin almashgandan keyingi holat.

Massiv qismlarini tartiblash “left_arrow > right_arrow” sharti o‘rinli bo‘lganidan so‘ng yakunlanadi. 9-rasmdagi holda bu shart “yolg‘on”, shuning uchun “right_arrow” chap tomonga qarab surilishda davom etmoqda. Bu holat 10-rasmdagi holat yuzaga kelguncha davom etadi.



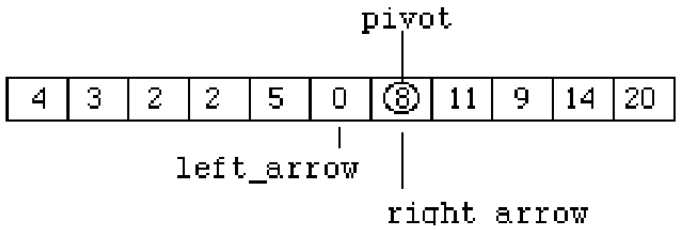
10-rasm. O‘ng tomonda almashish uchun element topildi.

“left_arrow”ning surilishi 11-rasmdagi holatga sabab bo‘ladi. O‘ng tomonga surilishda “pivot”dan katta yoki teng elementni topish talab qilingani uchun “left_arrow” chegaraviy elementga yetganidan keyin surishni to‘xtatadi.



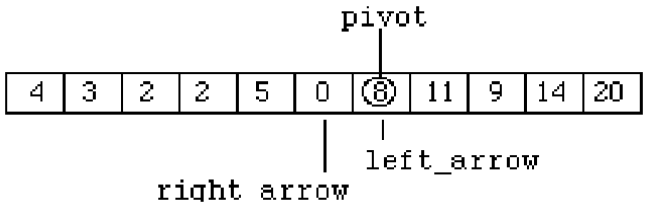
11-rasm. Chap element chegaraviy bilan ustma-ust tushdi.

Shundan keyin chegaraviy elementni o‘z ichiga oluvchi almashuv bajariladi va massiv 12-rasmdagi holatga o‘tadi.



12-rasm. Uchinchi almashuvdan keyingi holat.

Elementlar o‘rin almashganlaridan so‘ng “right_arrow” chap, “left_arrow” esa o‘ng tomonga suriladi (13-rasm).



13-rasm. Tartiblash indekslar massiv o‘rtasidan o‘tganda yakunlanadi.

13-rasmda tartiblash sharti “left_arrow>right_arrow” o‘rinli bo‘lgan hol tasvirlangan. Shuning uchun massivni ikkiga ajratish va qayta tartiblash jarayonini tugatilgan deb hisoblash mumkin.

Quyida tez tartiblash algoritmi *QuickSort* ni amalga oshiruvchi funksiya keltirilgan.

```
void quick_sort( int list[], int left, int right )
{
int pivot, left_arrow, right_arrow;
left_arrow = left;
right_arrow = right;
pivot = list[(left + right)/2];
do {
while ( list[right_arrow] > pivot )
right_arrow--; while ( list[left_arrow] < pivot )
left_arrow++; if ( left_arrow <= right_arrow )
{
swap( list[left_arrow], list[right_arrow] );
left_arrow++;
right_arrow--; }
 }
 while ( right_arrow >= left_arrow );
if ( left < right_arrow )
quick_sort( list, left, right_arrow );
if ( left_arrow < right )
quick_sort( list, left_arrow, right );
}
```

Takrorolash uchun savol va topshiriqlar

1. Rekursiya nima?

2. Rekursiv “cho‘kish” va “suzib chiqish” deganda nimani tushunasiz? Misollar bilan iizohlang.

3. Rekursiyaning afzalliklarini nimada deb o‘ylaysiz?

4. Quyidagi masalalar uchun dastur ishlab chiqing.

a) a^n ni eng kam ko‘paytirish amallari yordamida hisoblang;

b) faqat 1 sonini qo‘shish orqali ikkita natural sonni qo‘shing;

c) faqat qo‘shish amalidan foydalanib, ikki natural sonni ko‘paytiring;

d) rekursiyadan foydalanib hisoblang:

$$\sqrt{6+2\sqrt{7+3\sqrt{8+4\sqrt{9+...+n\sqrt{n+5}}}}}$$

e) Fibonachchi sonlari ketma-ketligi $f_0=1$, $f_1=1$, $f_{n+2}=f_{n+1}+f_n$, $n\geq 1$ formulalar bilan aniqlanadi. Shu ketma-ketlikning dastlabki k ta hadini rekursiv usulda toping.

9-§. KO‘RSATKICH VA XOTIRA BILAN ISHLASH

9.1. Boshlang‘ich tushunchalar

Unchalik murakkab bo‘lmagan dasturlarni ishlab chiqish uchun biz hozirgacha ko‘rgan ma’lumotlar yetarli, ammo murakkab dasturlar uchun kamlik qiladi.

Dasturchi murakkab kompyuter dasturlarini ishlab chiqishda quyidagi ma’lumotlarga ham javob berishi lozim:

- ma’lumotlar qayerda saqlanadi?
- u yerda qanday qiymatlar saqlanmoqda?
- saqlanayotgan ma’lumotlarning ko‘rinishi.

Bu masalalarga javob variantlaridan birinchisi o‘zgaruvchilarni odduy usulda e’lon qilishni nazarda tutadi. E’lon qilish buyrug‘i orqali ma’lumotning tipi va nomi ko‘rsatiladi. Kompilyator bu ma’lumot uchun kompyuter xotirasidan maxsus joy ajratishni ta’minlaydi va ichki vositalar yordamida uning manzilini nazorat qiladi.

Koʻrsatkichlardan foydalanishni nazarda tutuvchi ikkinchi variant barcha dasturchilar uchun oʻta muhim ahamiyat kasb etadi.

Koʻrsatkichlar ham oʻzgaruvchi boʻlib, oʻzida qandaydir qiymatlarni emas, balki bu qiymatlarning kompyuter xotirasidagi manzilini saqlaydi.

Koʻrsatkichlarni koʻrib chiqishdan avval oddiy oʻzgaruvchilar manzilini aniqlashga urinib koʻraylik. Agar ***home*** oʻzgaruvchi boʻlsa, ***&home*** bu oʻzgaruvchining kompyuter xotirasidagi manzilini anglatadi. Quyidagi listingga e'tibor bering:

```
# include <iostream.h>
int main()
{
int a=12;
double b=1.234;
cout<<a<<" a ning manzili "<<&a<<"\n";
cout<<b<<" b ning manzili "<<&b<<"\n";
return 0;
}
```

Ushbu kod Windows XP2 operatsion tizimi uchun quyidagi natijani ekranga uzatadi:

```
12  a ning manzili 0x8f7dfff4
1.234 b ning manzili 0x8f7dffec
```

Oʻzgaruvchilar manzilini ekranga uzatishda *cout* oʻn oltilik sanoq sistemasini qo'llaydi, chunki kompilyator xotira manzillarini saqlash uchun aynan shu sistemadan foydaladi. Bizning misolimizda *a* oʻzgaruvchining manzili *b* nikiga qaraganda 4 baytga farq qiladi. Buning sababi *int* tipidagi ma'lumotlarni saqlash uchun xotiradan 4 bayt joy ajratilishi bilan bogʻliq. Agar xotiraga *b* avval yozilsa, bu oʻzgaruvchilarning manzillari oʻrtasidagi farq 8 ga teng boʻladi. Chunki *double* tipidagi ma'lumotni saqlashga xotiradan 8 bayt joy band qilinadi.

C++ dasturlash tili Pascal kabi koʻplab tillardan shunisi bilan farq qiladiki, kompyuter tomonidan qandaydir qarorlar dastur kodini kompilatsiya qilish vaqtida emas, balki bajarish vaqtida, yaʼni dastur ishlayotgan vaqtda qabul qilinadi. Qiyoslash uchun misol keltiramiz: siz taʼtil vaqtida bevosita, vaziyatdan kelib chiqqan holda ob-havo, kayfiyat va imkoniyat kabi holatlarni hisobga olib, dam olishni tashkil qilishingiz mumkin. Bu bajarish vaqtida qabul qilinadigan qaror boʻladi. Kompilatsiya vaqtida qabul qilinadigan qarorga esa hech qanday vaziyatni eʼtiborga olmagan holda, belgilangan qatʼiy grafik asosida dam olishni misol qilib keltirish mumkin.

Bajarish vaqtida qabul qilinadigan qarorlar dasturning yuzaga keladigan vaziyatlarga moslashuvchan boʻlishini taʼminlaydi. Massivlar uchun xotirani taqsimlash masalasi bunga yaqqol misol boʻlishi mumkin. Odatda massivni eʼlon qilishda uning hajmi qatʼiy belgilab qoʻyiladi (kompilatsiya vaqti qabul qilinadigan qaror) va bu hajmni zarurat boʻlganda oʻzgartirish mumkin emas. Odatda, talabalar guruhi massivi bilan ishlaganda 25 ta element yetarli hisoblanadi. Ammo ayrim hollarda talabalar soni 15 ta yoki 30 yoki undan ham koʻp boʻlishi mumkin. Bunday hollarda har ehtimolga qarshi 50 ta elementli massiv eʼlon qilish lozim boʻladi. Qaralayotgan vaziyatda talablar soni 15 ta boʻlsa, xotiraning qolgan 35 yachaykasi boʻsh qolmoqda. Koʻrinib turibdiki, bunday dasturlar bilan ishlaganda xotiradan notoʻgʻri foydalanish muammosi yuzaga keladi. Bunday muammolarni (yaʼni xotiradan qancha kerak boʻlsa, shuncha joy ajratish masalasi kabi) hal qilish uchun C++ tilida qarorni dasturning bajarilishi jarayonida qabul qilish imkoniyati nazarda tutilgan. Shunday qilib, dastur ishga tushganda, kompyuter xotirasidan ehtiyojga koʻra zarur hajmni band qilishga koʻrsatma berish mumkin.

Xotiradagi maʼlumotlarni qayta ishlashning yangi strategiyasiga koʻra, qiymatlar joylashuvi haqidagi axborot nomlangan kattalik, qiymatning oʻzi esa hosila kattalik sifatida qayd qilinadi. Bunda maʼlumotlarning maxsus tipi – koʻrsatkichlardan foydalaniladi. Yuqorida aytib oʻtilganidek, ular qiymat saqlanayotgan xotira yacheykalari manzillarini oʻzida saqlaydi.

Yondosh qiymat (yoki qayta nomlash) deb ataluvchi amal yordamida koʻrsatilgan manzildagi qiymatni aniqlash mumkin. Quyidagi listingga e'tibor bering.

```
# include <iostream.h>
int main()
{
int a=12;
int *p_a;
p_a=&a;
cout<<a<<" a ning manzili "<<&a<<"\n";
cout<<*p_a<<"\n";
*p_a=*p_a+1;
cout << *p_a;
return 0;
}
```

Ushbu kod quyidagi ma'lumotlarni ekranga uzatadi:

```
12 a ning manzili 0x8fccfff4
12
13
```

Koʻrinib turibdiki, *a* oʻzgaruvchi tipi *int* hamda koʻrsatkich-oʻzgaruvchi *p_a* bitta tanganing ikki tomoni hisoblanadi. *P_a* koʻrsatkich *a* ni koʻrsatgani uchun, **p_a* va *a* lar ekvivalent. **p_a* ifodadan *int* tipidagi ma'lumot sifatida foydalanish mumkin. Listingdan koʻrinib turibdiki, unga qiymat ham berish mumkin. Bunda u koʻrsatayotgan oʻzgaruvchining qiymati ham toʻgʻridan-toʻgʻri oʻzgaradi.

9.2. Koʻrsatkichlarni e'lon qilish va initsializatsiya qilish

Kompilyator koʻrsatkich koʻrsatayotgan ma'lumot tipini nazorat qilib borishi lozim. Masalan, *char* tipidagi oʻzgaruvchining manzili ham *double* tipidagi oʻzgaruvchining manzili kabi boʻladi. Ammo bu tiplar uchun xotiradan turli kattalikdagi joy ajratiladi va ma'lumotlarni

saqlash uchun turli o‘lchamlar qo‘llanadi. Shuning uchun ko‘rsatkichlarni e’lon qilishda ular qanday tipdagi ma’lumotlarni ko‘rsatishi aniq belgilab qo‘yish lozim.

Oxirgi listingda * *p_a* ifodasi *int* tipida ekanligi ko‘rsatilgan. * amali ko‘rsatkichga nisbatan qo‘llangani uchun, *p_a* o‘zgaruvchining o‘zi ko‘rsatkich bo‘lishi lozim.

Eslatma. *p_a* ko‘rsatkich (manzil), **p_a* esa *int* tipidagi o‘zgaruvchi, ammo ko‘rsatkich emas.

Shuni ta’kidlash kerakki, * amalida bo‘sh joylardan foydalanish shart emas, ya’ni *int *a; int * a; int* a* e’lonlari ekvivalent hisoblanadi.

Boshqa tipdagi o‘zgaruvchilarga ko‘rsatkichlar ham aynan shu usul bilan amalga oshiriladi:

*double * ptr;* // *ptr* ko‘rsatkichi *double* tipini ko‘rsatmoqda;

*char * str;* // *str* ko‘rsatkichi *char* tipini ko‘rsatmoqda;

Ptr ko‘rsatkichi *double* tipini ko‘rsatgani uchun kompilyator **ptr* ning qiymati ham *double* bo‘lishini nazorat qiladi.

Ko‘rsatkichlarni initsializatsiya qilish uchun e’lon qilish operatoridan foydalanish mumkin. Bu holda faqat o‘zgaruvchi qayd qilinadi, uning qiymati esa ko‘rsatilmaydi:

*int a = 5; int * p_a = &a;*

buyrug‘i *p_a* ko‘rsatkichga (**p_a* ga emas) *&a* qiymat berilishini ta’minlaydi. Quyidagi listingga e’tibor bering.

```
# include <iostream.h>
int main()
{
int a=12;
int *p_a;
p_a=&a;
cout<<“a ning qiymati=“<<a<<” a ning manzili “<<&a
<<“\n”;
cout<<”*p_a ning qiymati=“<<*p_a<<”p_a ning qiymati=
```

“<<p_a<“\ n” ;
return 0;
}

U quyidagi natijani ekranga uzatadi:

a ning qiymati=12 a ning manzili 0x8fcefff4
*p_a ning qiymati=12 p_a ning qiymati=0x8fcefff4

Shuni yodda tutish lozimki, C++ tilida ko‘satkichlar e’lon qilinganda kompyuter xotirasidan manzil ko‘rsatayotgan ma’lumotlarni saqlash uchun emas, balki manzillarni saqlash uchun joy ajratadi. Ma’lumotlarni saqlash uchun esa qo‘shimcha amallarni bajarish lozim bo‘ladi. Masalan:

*long * b;*

buyrug‘i yordamida *long* tipidagi o‘zgaruvchiga ko‘rsatkich e’lon qilingan bo‘lsa,

*b=234567;

ko‘rsatmasini bajargan kompilyator kompyuter xotirasining ***b** manzilida 234567 sonini saqlab qo‘yadi.

Eslatma. Ko‘rsatkichlardan unumli foydalanish uchun manzillarning to‘g‘ri va aniq bo‘lishiga harakat qiling.

Ko‘rsatkichlar turli arifmetik ifodalar, qiyoslash operatorlari tarkibida uchrashi mumkin va C++ tilida ular ustida bir qator amallarni bajarish nazarda tutilgan. Ko‘rsatkichlarga nisbatan inkrement (++), dekrement (--), butun sonni qo‘shish yoki ayirish, ikki ko‘rsatkich orasidagi farqni topish kabi amallarni qo‘llash mumkin. Ko’rsatib o’tilgan bu amallar odatiy amallardan o’ziga hosligi bilan farq qiladi.

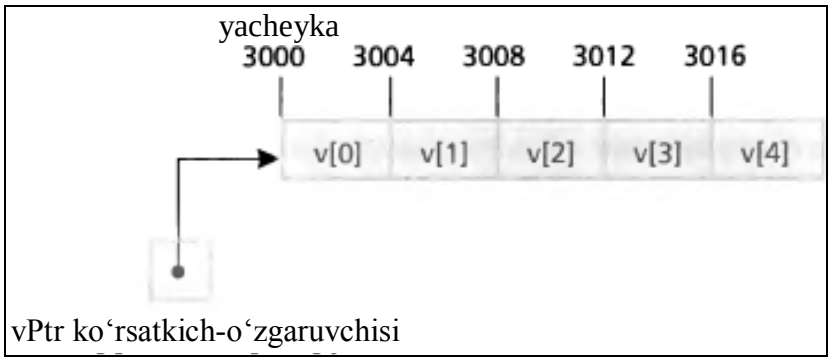
Faraz qilaylik, *int v*[10] massiv e’lon qilingan va uning birinchi elementi 3000-manzilga yozilgan bo‘lsin. Shuningdek, *vPtr* ko‘rsatkichi e’lon qilingan va unga *v*[0] ning manzili bo‘lgan 3000 ni yozib qo‘yilgan bo’lsin. Buning uchun

*int *vPtr=v;*

yoki

*int *vPtr=&v[0];*

buyruqlaridan birini yozish yetarli. Quyidagi rasmda ana shu holat tasvirlangan:



Odatiy arifmetikada 3000+2 amalining natijasi 3002 ga teng bo‘ladi. Ko‘rsatkichlar arifmetikasida esa bunday emas. Ko‘rsatkichga butun son qo‘shilganda yoki ayrilganda uning qiymati ko‘rsatkich ko‘rsatayotgan ma’lumot hajmi va ana shu son ko‘paytmasiga teng miqdorda o‘zgaradi. Masalan,

vPtr+=2; yoki *vPtr*=*vPtr*+2;

buyrug‘ining natijasi *vPtr* butun sonli (*int*) bo‘lgani uchun $3000+2\times4=3008$ ga teng bo‘ladi.

Ko‘rsatkich-o‘zgaruvchilarni bir-biridan ayirish ham mumkin. Faraz qilaylik, *vPtr* ko‘rsatkichi 3000, *v1Ptr* ko‘rsatkichi esa 3008 manzilni saqlab turgan bo‘lsin. U holda

$$x = vPtr - v1Ptr$$

buyrug’i bajarilganidan so‘ng, *x* ning qiymati 2 ga teng bo‘ladi. $((3008-3000)/4=2)$.

Odatda ko‘rsatkichlar ustida amallar bajarish massiv bilan ishlaganda uchrab turadi. Chunki massiv elementlari xotirada ketma-ket joylashadi va birinchi elementning manzili saqlanadi, xolos. Uning qolgan elementlariga ana shu manzilga element nomeri va ma’lumot hajmi ko‘paytmasini qo‘shib, murojaat qilinadi.

Ko‘rsatkichlar, agar ularning tiplari bir xil bo‘lsa, birining qiymatini boshqasi olishi mumkin. Aks holda tiplarni almashtirish buy-

ruqlaridan foydalangan holda ularni bir xil tipga keltirish talab qilinadi. Bundan faqat *void* * tipida koʻrsatkich istisno, xolos. *Void* tipidagi koʻrsatkich-oʻzgaruvchiga ixtiyoriy tipdagi koʻrsatkichning qiymatini berish mumkin. Ammo buning teskarisi mumkin emas.

9.3. *NEW* yordamida xotiradan joy ajratish

Umuman olganda, *oʻzgaruvchi* – xotiraning dasturni bajarish vaqtida nomlanadigan qismi hisoblanadi. Koʻrsatkichlar esa ana shu sohaning taxallusi sanaladi. Bu sohaga toʻgʻridan-toʻgʻri oʻzgaruvchining nomi orqali ham murojaat qilish mumkin.

Koʻrsatkichlarning afzalligi dasturni bajarish vaqtida xotiradan nomlangan sohalarni ajratib olishda (taqsimlashda) koʼrinadi. Dasturchilarga xotirani taqsimlash imkoniyatini ***new*** operatori taqdim etadi.

Bu operator qoʼllanganda xotiradan qanday tipdagi maʼlumot uchun joy ajratish lozimligi koʻrsatiladi. Bunda kompilyator xotiradan zarur hajmdagi blokni qidirib topadi va bu blok manzilini koʻrsatkichga oʻzlashtiradi.

Umumiy koʻrinishda *new* operatori quyidagicha yoziladi:

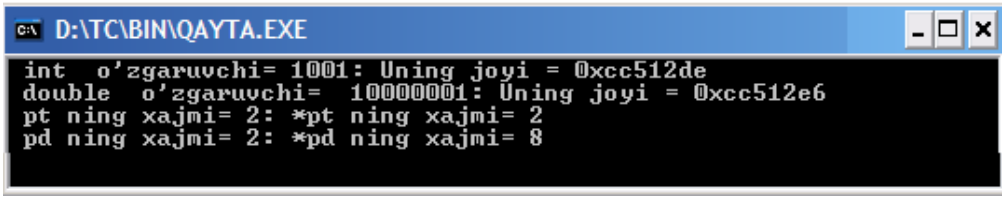
Tip_nomi Koʻrsatkich_nomi = new Tip_nomi ;

Maʼlumotlarning tipi ikki marta koʻrsatiladi: birinchi marta soʻralayotgan xotira turi belgilanadi, ikkinchi martasida esa zarur koʻrsatkich eʼlon qilinadi. Quyidagi listingga eʼtibor bering:

```
# include <iostream.h>
int main ()
{
int *pt=new int;       //int tipli ma'lumot uchun xotira ajratish
*pt=1001;              //qiymatni saqlash joyi
cout << " int " ;
cout << " o'zgaruvchi= "
<< *pt << ": Uning joyi = " << pt << "\n";
double * pd = new double;      // double uchun joy ajratish
*pd = 10000001.0;              // Qiymatni saqlash joyi
```

```
cout <<" double " ;
cout << " o'zgaruvchi= " << *pd <<": Uning joyi = "<< pd
<< "\n";
cout << " pt ning xajmi= "<< sizeof pt;
cout << ": *pt ning xajmi= " << sizeof *pt << "\n" ;
cout << " pd ning xajmi= " << sizeof pd;
cout << ": *pd ning xajmi= " << sizeof *pd << "\n";
return 0;
}
```

Ushbu dastur quyidagi natijani beradi:



Xotira umumiy hajmining chekli bo'lgani sababli, ko'plab nomlangan sohalarni ajratib olgan kompilyator vaqti kelganda navbatdagi *new* talabnomasini bajara olmay qolishi mumkin. Bunday vaziyatlarda *new* operatori 0 qiymatni qaytaradi. C++ tilida 0 qiymatli ko'rsatkich nolli ko'rsatkich (null pointer) deb ataladi. Undan asosan operator yoki funksiyalarning vazifalari buziladigan hollarda foydalaniladi.

Bunday holatlar yuzaga kelmasligi uchun xotiraning band qilingan va keyingi amallarda kerak bo'lmaydigan manzillarini bo'shatishga to'g'ri keladi.

9.4. Xotirani delete operatori yordamida bo'shatish

Xotiradan *new* yordamida band qilingan joylarni *delete* operatori bilan bo'shatish xotiradan unumli foydalanishga imkon beradi. Bo'shatilgan xotira bloklarini dasturning boshqa amallarida takroran ishga solish mumkin.

Delete operatori odatda *new* bilan birga qo'llanadi va umumiy ko'rinishda quyidagicha yoziladi:

*Tip_nomi * o‘zgaruvchi = new Tip_nomi;* // joy ajratish
. . . // xotiradan foydalanish
delete o‘zgaruvchi ; // o‘zgaruvchi band qilgan joyni bo‘shatish

Bu holda *o‘zgaruvchi* uchun xotiradan band qilingan joy bo‘shatiladi, ammo bu o‘zgaruvchining o‘zi yo‘qotilmaydi. Undan xotiraning boshqa qismini band qilish uchun foydalanish mumkin. Shuni yodda tutish lozimki, *new* va *delete* operatorlaridan muvozanatni saqlagan holda foydalanish lozim, aks holda dasturchi xotiraning ma’lum bir qismini “yo‘qotib qo’yishi” mumkin. Agar yo‘qotilgan xotira o‘ta katta bo‘lsa, kompilyator kompyuter xotirasidan zarur hajmdagi bo‘sh joyni topa olmagani sababli, dastur osilib qoladi.

Xotiraning bir blogiga ikkita ko‘rsatkich tashkil qilib bo‘lmaydi, chunki bu holda bu blokni bo‘shatishga ikki marta urinish ehtimolligi yuzaga keladi.

9.5. NEW operatori yordamida dinamik massivlarni tashkil qilish

Agar dastur faqat bitta ma’lumot bilan ish olib borsa, soddagina qilib bitta o‘zgaruvchini e’lon qilish mumkin. Ammo ko‘pincha katta sondagi ma’lumotlarni (masalan, massivlarni) qayta ishlashga to‘g‘ri kelganda bunday o‘zgaruvchilar dasturchi ishini murakkablashtirib yuborishi mumkin.

Faraz qilaylik, bajarish vaqtida olingan ma’lumotga ko‘ra massivdan foydalanish kerak bo‘lishi ham, bo‘lmasligi ham mumkin bo‘lgan dastur ishlab chiqilayotgan bo‘lsin. Bu muammoni *new* operatori yordamida hal qilish mumkin. U zarur bo‘lganda massiv yaratadi yoki aks holda yaratmaydi. Bundan tashqari, massiv o‘lchamini ham bajarish vaqtida aniqlash (tanlash) mumkin. Bajarish vaqtida yaratilgan massivlarni dinamik deb ataladi. Kompilatsiya vaqtida yaratiladigan massivlar esa statik massiv deyiladi.

C++ tilida dinamik massivlarni e’lon qilish uchun *new* operatoridan keyin massiv elementlarining tipi va kvadrat qavslar ichida miqdorini ko‘rsatish yetarli. Masalan,

*int * ps = new int* [10]; // *int* tipidagi 10 ta elementli blok

Bu holda *new* operatori ajratilgan blokning birinchi elementi manzilini **ps* ko‘rsatkichga o‘zlashtiradi.

Agar massivni yaratish uchun *new* operatoridan foydalanilgan bo‘lsa, u holda ajratilgan joyni bo‘shatish uchun *delete* operatori quyidagicha yoziladi:

delete [ ] ps; // dinamik massiv band qilgan joy bo‘shatiladi

Bu buyruqdagi kvadrat qavslar massivning alohida elementi band qilgan joyni emas, balki massiv to‘laligicha band qilgan joyni bo‘shatishni ta’minlaydi. Agar *new* operatori kvadrat qavslarsiz qo‘llangan bo‘lsa, u holda *delete* ham bunday qavslarsiz yoziladi.

Quyidagi ko‘rsatmalarga e’tibor bering:

```
int * pi = new int;
short * ps = new short [500];
delete [ ] pi;          // natija bermaydi, bunday yozmang
delete ps;              // natija bermaydi, bunday yozmang.
```

Shuni ta’kidlash joizki, *ps* ko‘rsatkichi faqat bitta *int* tipidagi elementni ko‘rsatadi. Blokdagi qolgan elementlarni boshqarish dasturchining vazifasi hisoblanadi. Kompilyator dastlabki 10 ta butun tipdagi elementlarni ko‘rsatishni nazorat qilmaydi, shuning uchun elementlar sonini dasturda e’tiborga olish talab qilinadi.

Massiv elementlari uchun xotiradan joy ajratish amali umumiy holda quyidagicha yoziladi:

tip_nomi ko‘rsatkich_nomi = new tip_nomi [elementlar_soni];

Natijada xotiradan *elementlar_soni* ta elementlarni saqlash uchun yetarli joy ajratiladi va *ko‘rsatkich_nomi* ko‘rsatkichiga bu joyning birinchi manzili o‘zlashtiriladi. Masalan,

*int *ps=new int [10];* // 10 ta butun sonli blok hosil qilinadi.

Bu yerda ko‘rsatkich 10 elementli blokning birinchi elementi manzilini ko‘rsatadi. Bu holatni ko‘rsatkich barmoq birinchi elementni ko‘rsatishiga qiyoslash mumkin. Navbatdagi elementni ko‘rsatish uchun bu barmoqni kerakli yo‘nalishda to‘rt baytga surish lozim bo‘ladi.

Birinchi elementga murojaat qilish muammo emas. Uning qolgan elementlariga qanday murojaat qilish mumkin? Ma’lumki, **ps* birinchi element qiymatini saqlaydi. Uni soddaroq qilib, *ps[0]* tarzida ham yozish mumkin. Shuning uchun ikkinchi elementga *ps[1]* orqali murojaat qilinadi. Qolgan elementlarni ham shu tariqa ko‘rsatish mumkin.

Quyidagi dasturga e’tibor bering.

```
#include  <iostream.h>
using namespace std
int main()
{
double * r3 = new double [3] ;   // uch element uchun joy
r3[0] = 0.2;                      // r3 massiv nomi sifatida talqin
r3[1] = 0.5;                      // qilinadi
r3[2] = 0.8;
cout << “r3[1] = “ <<r3[1] <<”\n”;
r3 = r3 + 1;                      // ko‘rsatkich orttirilmoqda
cout <<”Yangi r3[0] = “ <<r3[0] <<” hamda”;
cout << “r3[1] = “ << r3[1] << “\n”;
r3 = r3 – 1;                          // blok boshini ko‘rsatadi
delete [] r3;                         // xotirani bo‘shatish
return 0 ; }
```

Bu dastur quyidagi natijani beradi:

p3[l] = 0.5
Yangi p3[0] = 0.5 hamda p3[l] = 0.8

Massiv nomini o‘zgartirib bo‘lmaydi, ammo ko‘rsatkich o‘zgaruvchi bo‘lgani uchun uning qiymatini o‘zgartirish mumkin. Yuqoridagi listingdan ko‘rinib turibdiki, *p3* ga 1 ni qo‘shilganidan so‘ng, *p3[0]* ifoda ilgari ikkinchi hisoblangan elementni ko‘rsata boshlaydi. Shuning uchun *delete [ ]* yordamida xotirani to‘g‘ri bo‘shatish uchun *p3* dan 1 ayrilmoqda.

Massiv va ko‘rsatkichlar o‘rtasidagi aloqalarni satrlar uchun ham tatbiq etish mumkin. Quyidagi buyruqlarga e’tibor bering:

char gul[10] = “atirgul”;
cout <<gul <<”qizil\n”;

Ma’lumki, massiv nomi birinchi element manzilini anglatadi. Shuning uchun *cout* operatoridagi *gul* o‘zgaruvchisi *char* tipida bo‘lib, birinchi harf (“*a*” harfi) manzilini ko‘rsatadi.

Cout buyrug‘i *char* tipidagi massiv manzilini satrning ham manzili deb hisoblaydi va to satrning oxirgi belgisi bo‘lgan nol belgi (\0) gacha bo‘lgan barcha belgilarni ekranga chiqaradi.

Agar *cout* operatorida belgining manzili ko‘rsatilsa, u holda shu belgidan boshlab barcha belgilarni ekranga uzatadi.

9.6. *CONST* tipidagi ko‘rsatkichlar

Ma’lumki, *const* kalit so‘zi bilan belgilangan ma’lumotlar haqida kompilyatorga mazkur ma’lumotning o‘zgarmas ekanligi ta’kidlanadi. Ko‘rsatkichlar uchun ham bu mulohaza o‘rinli.

Ko‘rsatkichlarni e’lon qilishda *const* xizmatchi so‘zi tip spetsifikatoridan yoki oldin, yoki keyin yozish mumkin. Masalan, ularni quyidagi variantlardan birida e’lon qilish to‘g‘ri hisoblanadi:

*const int *pOne;*
*int * const pTwo;*
*const int * sonst pThree;*

Bu misolda *pOne* ko‘rsatkichi *int* tipidagi o‘zgarmas manzilni o‘zida saqlaydi. Shuning uchun u ko‘rsatayotgan qiymatni o‘zgartirish mumkin emas.

pTwo ko‘rsatkichi – *int* tipli ko‘rsatkich bo‘lgan o‘zgarmasni anglatadi. Bu holda ko‘rsatkich manzilida yozilgan qiymatni o‘zgartirish mumkin, lekin adresning o‘zini o‘zgartirib bo‘lmaydi.

Nihoyat, *pThree* ko‘rsatkichi - *int* tipidagi o‘zgarmas manzilini o‘zida saqlovchi o‘zgarmas ko‘rsatkich. U xotiraning faqat bitta manzilini o‘zida saqlaydi, bu manzildagi qiymat o‘zgarmas ham, o‘zgaruvchi ham bo‘lishi mumkin.

Birinchi navbatda qanday qiymat o‘zgarmas deb e’lon qilinayotganligini bilish kerak. Ko‘rsatkichning o‘zimi yoki unda saqlanayot-

gan qiymatmi? Agar *const* kalit so‘zi o‘zgaruvchi tipidan oldin yozilgan bo‘lsa, demak, aniqlangan o‘zgaruvchi o‘zgarmas bo‘ladi. Agar *const* kalit so‘zi o‘zgaruvchi tipidan keyin qo‘yilgan bo‘lsa, u holda ko‘rsatkichning o‘zi ham o‘zgarmas bo‘ladi. Quyidagi misollarni qaraymiz.

```
Const int * p1    // int tipli o‘zgarmasga  ko‘rsatkich
Int *const p2     // o‘zgarmas ko‘rsatkich, ya’ni  doimo
                  // xotiradagi bir sohani ko‘rsatib turadi.
```

Masala. Ko‘rsatkichlar yordamida bir o‘lchovli massiv elementlarini o‘sish tartibida qayta tartiblang.

Yechish g‘oyasi. Massiv elementlarini tartiblash g‘oyasi 6.2-§da keltirilgan va biz bu haqda to‘xtalib o‘tmaymiz hamda asosiy e’tiborni ko‘rsatkichlardan foydalanishga qaratamiz.

Massiv o‘lchami funksiyada yuqori indeksni ko‘rsatish uchun xizmat qiladi va funksiya jismida o‘zgarmas bo‘ladi. Shuning uchun uni o‘zgarmas deb e’lon qilish zarur. Massiv elementlari shunchaki chop qilingani uchun uni ham o‘zgarmas deb e’lon qilish mumkin.

Funksiyaga ko‘rsatkichlarni uzatishning to‘rtta usuli mavjud:

1) o‘zgaruvchi ko‘rsatkichlarni o‘zgaruvchi ma’lumotlarga;

2) o‘zgaruvchi ko‘rsatkichlarni o‘zgarmas ma’lumotlarga;

3) o‘zgarmas ko‘rsatkichlarni o‘zgaruvchiga;

4) o‘zgarmas ko‘rsatkichlarni o‘zgarmas ma’lumotlarga.

Bu usullarning farqi shundaki, o‘zgaruvchi ko‘rsatkich va ma’lumotlarni o‘zgartirish mumkin va odatda ularni e’lon qilishda *const* kalit so‘zi ishlatilmaydi.

Ushbu masalaning dasturi quyidagicha yoziladi[2]:

```
# include <iostream.h>
# include <iomanip.h>
void selectionSort(int * const, const int);
void swap (int * const, int * const);
int main()
```

[2] Дейтел Х. М., Дейтел П.Дж. Как программировать на C++. – М.: “Бином”, 2008. -С. 525.

```
{ const int arraysize=10;
int a[arraysize]={2, 6, 4, 8, 10, 12, 89, 68,45, 37};
cout <<“Massiv elementlari tartiblashdan avval \n”;
for (int i=0; i<arraysize; i++)
cout <<setw(4)<<a[i];
selectionSort (a, arraysize);    // massivni tartiblash
cout <<“Massiv elementlari tartiblashdan so‘ng \n”;
for (int j=0; j<arraysize; j++) ;
cout <<setw(4)<<a[j];
cout <<endl1;
}
void selectionSort( int * const array, const int size)
{
int smallest ;                    // eng kichik indeks nomeri
for (int i=0; i<size-1; i++)
{ smallest=i;                     // tartiblanayotgan element nomeri
 for (int index=i+1; index<size; index++)
if (array[index]<array[smallest])
smallest=index;
swap(&array[i], &array[smallest]);
}          //  if  tugadi
}          //  selectionSort funksiyasi  tugadi
void swap(int *const element1Ptr, int *const element2Ptr)
{
int hold=*element1Ptr;
*element1Ptr=*element2Ptr;
*element2Ptr=hold;
 }      // swap tugadi.
```

Dastur kodini tekshirish o‘quvchilarga havola qilinadi.

9.7. *Main()* funksiyasi haqida

Dastur ishga tushganidan keyin boshqaruv uzatiladigan funksiyaning nomi *main* bo‘lishi lozim. U o‘ziga murojaat qilgan tizimga qiymatlarni qaytarishi yoki tashqi muhitdan qabul qilishi mumkin.

Qaytariladigan ma’lumotlarning tipi butun bo‘ladi. Bu funksiyani dasturchi zaruratga qarab, dasturning ixtiyoriy qismida keltirishi mumkin.

Main() funksiyasini ikki xil ko‘rinishda yozish qabul qilingan:

```
// parametrlarsiz:
tip main()
{ /* ... */}

// ikkita parametr bilan:
tip main(int argc, char* argv[])
{ /* ... */}
```

Dastur buyruqlar satridan ishga tushirilganda parametrlar bir-biridan bo‘sh joy belgisi bilan ajratib yoziladi. Dasturda parametrlarning nomlari turlicha bo‘lishi mumkin, ammo odatda ularni *argc* va *argv* bo‘lishi tavsiya etiladi. Bu parametrlarning birinchisi *argc* funksiyaga uzatilishi rejalashtirilgan parametrlar sonini ko‘rsatadi. Ikkinchisi esa *char** tipidagi ko‘rsatkichlar massiviga ko‘rsatkich hisoblanadi. Bu massivning har bir elementi nol-simvol bilan tugaydigan buyruqlar satrining alohida parametriga ko‘rsatkichni saqlaydi. Massivning birinchi elementi *argv[0]* ishga tushirilayotgan dasturning to‘liq nomiga ishora qiladi. Keyingi *argv[l]* element birinchi parametrga, *argv[2]* ikkinchi parametrga va h.k. tarzda ma’lumotlarga ko‘rsatadi. *Argv[argc]* parametr 0 ga teng bo‘lishi lozim.

Main () funksiyasining parametrlari dasturga buyruqlar satrida keltirilgan (dastur ishga tushganidan keyin klaviaturadan kiritiladigan) ma’lumotlarni uzatish uchun xizmat qiladi.

Agar *main()* funksiyasi hech qanday qiymatni qaytarmasa (*void* tipi), unga murojaat qilgan tizim ishni muvafaqqiyatli tugatilganligi haqida axborotni (qiymatni) qabul qiladi. Nol bo‘lmagan qiymat esa dasturning kutilmaganda (halokatli) tugatilganini anglatadi. Buyruqlar satrining maksimal uzunligi bo‘sh joy belgilarini ham hisobga olganda 128 dan katta bo‘lmasligi lozim. Bu chegarani DOS qo‘yadi.

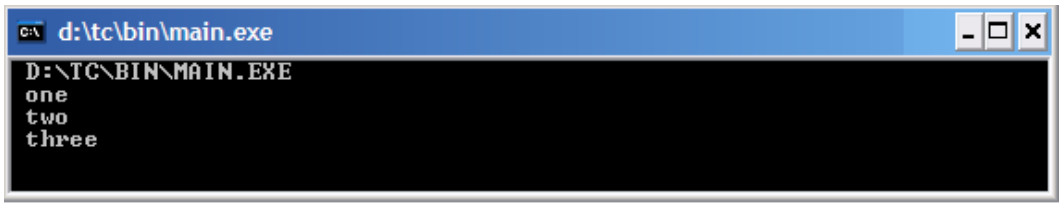
Quyida dasturni ishga tushirishda ko‘rsatilgan barcha parametrlarni qaytaruvchi dasturni ko‘rib chiqamiz.

```
#include <iostream.h>
void main(int argc, char* argv[])
{
for (int I = 0: i<argc; i++) cout << argv[i] << '\n';
}
```

Faraz qilaylik, ishga tushirilayotgan faylning nomi *main.exe* bo‘lsin va u bilan birga

d:\TC\BIN\MAIN.EXE one two three

parametrlari kiritilgan bo‘lsin. Bu holda ekranga quyidagi ma’lumotlar chiqariladi:



Ayrim hollarda bo‘sh joy belgisini ham parametr (masalan, biror jumlani bitta parametr) sifatida uzatish talab qilinishi mumkin. Bunday hollarda jumla qo‘shtirnoq belgilari bilan ko‘rsatiladi:

“C++ dasturlash tili”

Main() funksiyasi dastur o‘z ishini tugatganidan keyin maxsus kodli qiymatni (butun son) qaytaradi. Ammo agar dastur ishini *exit* (yoki *_exit*) funksiyasi yordamida tugatilgan bo‘lsa, ana shu funksiyaning qiymati qaytariladi. Masalan, dastur

exit(1);

ko‘rsatmasi yordamida o‘z ishini tugatgan bo‘lsa, funksiyaning kodi 1 qiymatiga teng bo‘ladi.

Takrorlash uchun savol va topshiriqlar

1. Ko‘rsatkich nima va undan qanday maqsadlarda foydalaniladi?
2. Xotiraning manzillarini qanday band qilish mumkin?
3. Ko‘rsatkich bilan band qilingan xotira qanday bo‘shatiladi?

4. Massivlar bilan ishlashda ko‘rsatkichlardan qanday foydalanish mumkin?

5. *Const* tipidagi ko‘rsatkichlardan foydalanish usullarini ayting.

6. Quyidagi masalalarni ko‘rsatkichlardan foydalanib hal qiling.

a) b o‘zgaruvchini e’lon qiling va uning xotiradagi o‘rnini aniqlang;

b) ko‘rsatkichlardan foydalanib, berilgan *a* va *b* o‘zgaruvchilar qiymatlarini almashtiring;

c) double tipidagi *a*[10] massivning birinchi elementi xotiraning 2000-yacheykasida saqlanayotgan bo‘lsin. Uning beshinchi elementi qaysi yacheykada saqlanadi?

d) s1 massivda saqlanayotgan satrni (matnni) *s2* massivga ko‘chiring.

10-§. FOYDALANUVCHI ANIQLAYDIGAN TIPLAR

10.1. Kirish

Dastur ishlab chiqish amaliyotida tez-tez hal qilinayotgan masala xarakteridan kelib chiqqan holda turli xil tip va murakkablikdagi ma’lumotlarni qayta ishlashga to‘g‘ri keladi. Agar bu ma’lumotlar yetarlicha murakkab tuzilmaga ega bo‘lsa, ularni standart usullar bilan ifodalash dasturchi ishining qiyinlashib ketishiga sabab bo‘lishi mumkin. C++ dasturlash tilida bunday ma’lumotlarni to‘g‘ri tavsiflash va qayta ishlash uchun oddiy tipdagi ma’lumotlardan foydalangan holda butunlay yangi tiplarini e’lon qilish va ular ustida turli amallarni bajarish ham nazarda tutilgan.

Bunday tipdagi ma’lumotlarni foydalanuvchi aniqlaydigan tiplar deb atash qabul qilingan. Quyida ana shunday tiplarning ayrimlari bilan tanishamiz.

10.2. Tiplarni qayta nomlash (typedef)

Dasturni o‘qish va tushunishni osonlashtirish uchun *typedef* buyrug‘i yordamida tiplarni yangi nomlar bilan atash mumkin. Bu buyruqning umumiy ko‘rinishi quyidagicha:

typedef tip yangi_nom [o‘lcham];

Bu holda kvadrat qavslar sintaksis elementi hisoblanadi. Zarur bo‘lmasa, o‘lchamni tushirib qoldirish mumkin. Masalan:

typedef unsigned int uint;
typedef char msg[100];
typedef stmet {char fio[30]; int date, code; double oylik;} ishchi;

Bu usul bilan kiritilgan tiplardan xuddi standart tiplar kabi foydalanish mumkin:

uint i, j; // *unsigned int* tipidagi ikkita o‘zgaruvchi
msg str[10]; // har birida 100 tadan belgisi bo‘lgan 10 ta satr
ishchi staff[100]; // 100 ta strukturadan iborat bo‘lgan massiv

Ko‘rinib turibdiki, *typedef* operatori uzundan-uzun nomli tiplarni qisqa va tushunarli qilib tavsiflash imkonini beradi.

10.3. Elementlari sanaladigan tiplar (enum)

Ko‘pincha dastur ishlab chiqish jarayonida nomlangan bir nechta o‘zgarmas ma’lumotlardan foydalanishga to‘g‘ri keladi. Bunda bu o‘zgarmaslarning har biri turlicha ma’nolarni anglatishi va hatto bu ma’lumotlarning asl ma’nosi buzilib ham ketishi mumkin.

Bunday hollarda elementlari sanaladigan yangi tiplarni e’lon qilish va foydalanish tavsiya etiladi. Bu ko’rsatma umumiy ko‘rinishda quyidagicha yoziladi:

enum tip_nomi {o‘zgarmaslar ro‘yxati};

Shundan keyin bunday tiplar bilan ishlash imkoniyati paydo bo‘ladi. Kompilyator esa o‘z navbatida bu tipdagi o‘zgaruvchilar faqat o‘zgarmaslar ro‘yxatida ko‘rsatilgan qiymatlarni qabul qilishini nazorat qiladi. Masalan:

Enum yangi {besh, olti, yetti, sakkiz};

ko‘rsatmasi yordamida faqat *besh, olti, yetti, sakkiz* qiymatlaridan iborat bo‘lgan ***yangi*** tipi e’lon qilindi. Bu holda *besh, olti, yetti, sakkiz* elementlari mos ravishda 0 dan boshlab 3 gacha bo‘lgan tartib raqamiga ham ega bo‘ladi. Namuna uchun sizga yaxshi tanish bo‘lgan masala uchun dastur kodini keltiramiz:

```
# include <iostream.h>
void main()
{
  enum medal {bir, ikki, uch, turt, besh} ;
  medal urin;
  urin=ikki;
  switch (urin)
  {
    case 0: cout<<"Oltin"; break;
    case 1: cout<<"Kumush"; break;
    case 2: cout<<"Bronza"; break;
    default: cout<<"Medal olmagan";
   }
 }
```

Bu yerda yuqorida aytilganidek, bir, ikki, uch, to‘rt, besh o‘zgarmaslarga mos ravishda 0, 1, 2, 3 va 4 qiymatlari berilgan. Ushbu dastur ishga tushirilsa, ekranga *kumush* yozuvi chiqariladi.

Agar zarur bo‘lsa, o‘zgarmaslarning qiymatlarini ko‘rsatib qo‘yish ham mumkin:

enum {two = 2, three, four, ten = 10, eleven, fifty=ten+40};

Bu yerda *three* va *four* o‘zgarmas nomlarga 3 va 4, *eleven* ga – 11, *fifty* ga esa 50 qiymati beriladi.

Elementlari sanaladigan o‘zgarmaslarning nomlari betakror bo‘lishi lozim.

Elementlari sanaladigan tiplar doirasida arifmetik amallarni bajarishda, o‘zgarmaslar songa aylantiriladi. Masalan, yuqoridagi dasturda

urin= ikki;

buyrug‘ini

*urin=ikki*2;*

bilan almashtirilsa, ekranga *bronza* yozuvi chiqariladi.

10.4. Strukturalar (struct)

Maktab o‘quvchisi tabelini ko‘z oldingizga keltiring. Unda viloyat, tuman nomi, maktab raqami, sinfi, o‘quvchining familiyasi, ismi, otasining ismi, o‘quv yili hamda o‘quvchining turli fanlar bo‘yicha chorak va yillik baholari saqlanadi. Demak, bitta sinfda 25 ta o‘quvchi bo‘lsa, ular haqidagi hamma ma’lumotlarni saqlash uchun 25 dona tabel zarur bo‘ladi. Endi har bir tabeldagi barcha ma’lumotlarni bitta satrga yozilgan holatini ko‘z oldingizga keltiring. Demak, 25 ta satrdagi ma’lumotlar jamg‘armasi hosil bo‘ladi. Bu jamg‘armadan ma’lumotlarni olish yoki yozish ishlarini osonlashtirish maqsadida mazmun jihatidan bir xil bo‘lgan ma’lumotlarni alohida ustunlarga yoziladi. Masalan, viloyatlar bitta ustunga, o‘quvchilarning familiyalari bitta, ismlari boshqa ustunga yoziladi.

Viloyat	Tuman	Sinf	Familiyasi va ismi
Namangan	Chortoq	7	Abdullayev Ilhom
Namangan	Uychi	11	Botirova Klara
Farg‘ona	Yaypan	10	Daminov Sodiqjon
Andijon	Bo‘z	9	Komilova Gulchehra
Namangan	Pop	7	Salimova Gavharoy
Farg‘ona	Qo‘qon	7	Tursunov Abdulla

Hosil bo‘lgan ustunlar maydon deb ataladi. Har bir satr esa yozuv deyiladi. Demak, yozuv maydonlar to‘plamidan iborat. Har bir maydon mazmun va tipi bir xil ma’lumotlarni o‘z ichiga oladi.

Ko‘rinib turibdiki, bir yozuvni bitta o‘zgaruvchi deb qarasak, u o‘zgaruvchi turli tipdagi qiymatlarni qabul qiladi va dastur ishlab chiqish jarayonini sezilarli darajada osonlashtiradi. Bunday o‘zgaruvchilarni C++ dasturlash tilida struktura deb yuritiladi. Ular umumiy ko‘rinishda

```
Struct  tip_nomi
{
1_tip 1_element;
```

2_tip 2_element;
....
n_tip n_element;
*} initsializator **;***

tarzida yoziladi. Masalan, yuqorida keltirilgan jadval uchun quyidagi strukturani e’lon qilish mumkin:

```
struct uquvchi
{
 char viloyat[10];
 char tuman[12];
 int sinf;
 char fio[30]
}
```

Struktura elementlari uning maydonlari deb ataladi.

Agar nom ko‘rsatilmagan bo‘lsa, u holda bu struktura *initsializator*da ko‘rsatilgan nom bilan ataladi va maydonlari qiymatlarni mos ravishda oladi:

```
 struct
       {
 char viloyat[10];
 char tuman[12];
 int sinf;
 char fio[30]
} uquvchi={“Namangan”, “Chortoq”, 7, “Abdullayev Ilhom”};
```

Struktura nomini e’lon qilinganidan keyin boshqa o‘zgaruvchilar kabi foydalanish mumkin.

Yuqorida aniqlangan struktura uchun

uquvchi a[25], b;

buyrug‘i har bir elementi *uquvchi* tipida (ya’ni, *viloyat[10], tuman[12], sinf* va *fio[30]* maydonlariga ega) bo‘lgan *a* massiv hamda *uquvchi* tipidagi *b* o‘zgaruvchilarni e’lon qiladi.

O‘zgaruvchilarning struktura maydonlariga murojaat qilish uchun

o‘zgaruvchining_nomi.maydon nomi

shaklidagi ko‘rsatmadan foydalaniladi. Masalan, *uquvchi* struktura-sidagi ma’lumotlarga (maydonlarga) *uquvchi.viloyat[], uquvchi.sinf, uquvchi.fio[]* tarzida murojaat qilish mumkin. Shundan keiyn, navbat-dagi barcha amallarda bu tipdagi ma’lumotlardan boshqa tipdagi amallar kabi foydalanish mumkin.

Masala. 25 ta talabaning familiyasi va ismi, informatika fanidan to‘plagan ballari berilgan bo‘lsin. Talabalarning umumiy o‘rtacha ballarini aniqlang. O‘rtachadan yuqori o‘zlashtirishga erishgan tala-balar haqidagi ma’lumotlarni ekranga chiqaring.

Yechish **g*oyasi.*** Dastlab *talaba* tipidagi struktura, so‘ngra shun-day tipdagi massiv e’lon qilinadi. Ma’lumotlar kiritilganidan keyin umumiy o‘rtacha ballar aniqlanadi. Massivni qayta ko‘rib chiqib, undan yuqori ball to‘plagan talabalar haqidagi ma’lumotlar ekranga chiqariladi. Dastur kodi quyidagicha:

```
# include <iostream.h>
void main()
{
struct talaba {char fish[20]; float bal;};
talaba inf[25] ;
int i; float s=0;
  for (i=0; i<25; i++)
  {
  cout<<i+1<<”–talaba familiyasi”;
cin>>inf[i].fish; cout<<”\n”;
  cout<<i+1<<”–talaba ballari
“; cin>>inf[i].bal; cout<<”\n”;
  s=s+inf[i].bal;
  }
  s=s/25;
  for (i=0; i<25; i++)
```

```
  if (inf[i].bal>s)
  cout<<inf[i].fish<<"   "<<inf[i].bal<<"\n";
}
```

Dasturni tekshirib ko‘rishni talabalarga havola qilamiz.

10.5. Bitli maydonlar

Bitli maydonlar – strukturalarning o‘ziga xos maydonlari sanalib, ma’lumotlarni ixcham qilib tavsiflash uchun xizmat qiladi. Odatda bunday maydonlar ha/yo‘q shaklidagi ma’lumotlar uchun tashkil qilinadi va ularni saqlash uchun xotiradan 1 bit joy yetarli bo‘ladi.

Bitli maydonlarni tavsiflashda nomdan keyin ikki nuqta yoziladi va maydon uzunligi bitlarda (butun musbat konstanta) ko‘rsatiladi.

```
struct Hodim
{
char fish[20];
bool jinsi:l:
}
```

Bitli maydonlarga ularning nomi orqali murojaat qilinadi va buyruqlarni yozishda boshqa maydonlar kabi foydalanish mumkin bo‘ladi.

10.6. Birlashmalar (union)

Birlashmalar (*union*) strukturalarning xususiy holi bo‘lib, barcha maydonlari bitta xotira adresida joylashadi. Birlashmalar umumiy ko‘rinishda xuddi struktura kabi e’lon qilinadi, faqat *struct* o‘rniga *union* yoziladi:

```
union  tip_nomi
{
1_tip 1_element;
2_tip 2_element;
....
n_tip n_element;
}
```

Birlashma uzunligi maydonlarining eng katta uzunligiga teng bo‘ladi. Har bir vaqt momentida birlashma tipidagi ma’lumotda faqat bitta ma’lumot saqlanadi, xolos. Bu ma’lumotdan to‘g‘ri foydalanish mas’uliyati dasturchi zimmasiga yuklatilgan.

Odatda birlashmalardan oldindan bir vaqtning o‘zida faqat bitta maydondan foydalanish zarur bo‘lgan hollarda xotirani tejash maqsadida tashkil qilinadi.

Birlashmalardan odatda struktura maydoni sifatida foydalaniladi. Bunda strukturaga aynan birlashmaning aynan qaysi elementi zarurligini ko‘rsatuvchi qo‘shimcha maydon kiritilishi mumkin. Birlashma nomini ko‘rsatish shart emas. Bu holat uning maydonlariga bevosita murojaat qilish imkonini beradi.

Quyidagi dastur kodini tahlil qiling.

```
# include <iostream.h>
int main()
{
enum paytype {card, check};
paytype ptype;
union payment
{
char card[25];
long check;
} info;
switch (ptype)
{
  case card: cout<<”kartochka bilian tulov  “<<info.card; break;
  case check: cout<<”naqd pul bilian tulov “<<info.check; break;
}
return 0;
}
```

Strukturalardan birlashmalar quyidagi jihatlari bilan farq qiladi:

– birlashmalar faqat birinchi elementi qiymati bilan initsializatsiya qilinadi;

– birlashmalar bitli maydonlarni o‘z ichiga olmaydi;
– birlashmalarda virtual metod, konstruktor va destruktor, qiymat berish amallaridan foydalanib bo‘lmaydi;
– birlashmalar klasslar shajarasiga kira olmaydi.

Takrorlash uchun savol va topshiriqlar

1. Tiplarni qayta nomlashning ahamiyatini tushuntiring.
2. Elementlari sanaladigan tiplar nima?
3. Strukturalar bilan qanday ishlash mumkin?
4. Bitli maydonlar qachon tashkil qilinadi?
5. Birlashmalar strukturadan asosan nimasi bilan farq qiladi?
6. Quyidagi masalalar uchun dastur ishlab chiqing.

a) Magazindagi 25 xil tovarning nomi va bahosi berilgan bo‘lsin. Magazindagi eng arzon va qimmat tovarlarni aniqlang.

b) Kunning o‘tib borayotgan soat, minut va sekundlari berilgan bo‘lsin. Tushlikkacha (12:00:00) qancha vaqt qolganini aniqlang.

c) 25 ta abonentning telefon kompaniyasi nomi, telefon raqamlari va familiyasi berilgan bo‘lsin. Berilgan familiyali abonent haqidagi ma’lumotlarni ekranga chiqaring.

11-§. MA’LUMOTLARNING DINAMIK STRUKTURALARI

11.1. Boshlang‘ich ma’lumotlar

Har qanday dastur ma’lumotlarni qayta ishlash uchun mo‘ljallangan bo‘lib, uning algoritmi ko‘pincha ma’lumotlarni tashkil qilinishiga to‘g‘ridan-to‘g‘ri bog‘langan bo‘ladi. Shuning uchun ma’lumotlarning strukturasi algoritmlardan avval tanlanishi lozim. Biz hozirgacha ma’lumotlarning standart tiplari bilan ishlab keldik. Ko‘rdikki, dasturlarda massivlar va strukturalar tez-tez uchrab turadi va odatda qanday hajmdagi ma’lumotlarni qayta ishlash oldindan ma’lum bo‘lmaydi.

Agar ish boshlagan dasturda zarur bo‘lgan ma’lumotlarni saqlash uchun qancha xotira zarur bo‘lishini aniqlashning iloji bo‘lmasa, xotira ehtiyojga ko‘ra bir-biri bilan ko‘rsatkichlar orqali bog‘langan alohida bloklar yordamida ajratiladi. Ma’lumotlar tashkil

qilishning bunday usuli ***ma'lumotlarning dinamik strukturalari*** deb ataladi, chunki bu ma'lumotlarning o'lchami dasturning bajarilishi davomida o'zgarib turadi.

Dastur ishlab chiqishda chiziqli (bir tomonlama) ro'yxat, stek, navbat va binar daraxt deb ataluvchi dinamik strukturalardan keng foydalaniladi. Bu ma'lumotlar bir-biridan tashkil qilish usuli va ruxsat etilgan amallari bilan farqlanadi. Dinamik strukturalar xotiraning qo'shni bo'lmagan qismlarini band qilishi mumkin.

Dinamik strukturalar o'lchami ma'lum, ammo katta hajmli ma'lumotlar bilan ishlashda muhim ahamiyat kasb etadi. Masalan, elementlar soni yetarlicha katta bo'lgan massivni tartiblashda ma'lumotlarning o'rinlarini almashtirishga qaraganda bu elementlarga ko'rsatkichlar almashtiriladi. Buning evaziga dasturning ishlash tezligi bir necha marta ortishi mumkin. Massivning biror elementini qidirishda binar daraxtlardan foydalanish ham dasturning ishlash tezligini sezilarli darajada orttiradi.

Ixtiyoriy dinamik strukturali ma'lumotlarning o'zi ham struktura bo'lib, kamida ikkita maydon (ma'lumotni saqlash va ko'rsatkichlar uchun) yoki undan ham ko'p bo'lishi mumkin. Ma'lumot va ko'rsatkichlarning maydonlari bir nechta bo'lib, ma'lumotlar maydoni asosiy (tarkib) tiplaridan birida yoki ko'rsatkich tipida bo'lishi mumkin. Eng sodda elementni tavsiflash quyidagicha ko'rinishda bo'ladi:

```
struct Node{
Data d;              // Data tipi Node dan oldinroq e'lon qilinadi
Node *p:
};
```

Ma'lumotlarning dinamik tiplarini bayon qilishga o'tamiz.

11.2. Chiziqli (bir tomonlama) ro'yxatlar

Elementlarni bir-biri bilan bog'lashning eng sodda usuli – bu har bir elementning o'zidan keyingi element bilan bog'lanishidir. Ma'lumotlarni bunday usulda bog'lanishi bir tomonlama ro'yxat deb ataladi.

Agar buning ustiga har bir element o‘zidan oldingisi bilan ham bog‘-langan bo‘lsa, bunday bog’lanish ikki tomonlama ro‘yxat deyiladi. Agar ikki tomonlama ro‘yxatning oxirgi elementi ro‘yxatning birinchi (bosh) elementi bilan bog‘lansa, halqali ro‘yxat hosil bo‘ladi.

Ro‘yxatning har bir elementi bu elementni boshqalaridan ajratib turuvchi kalitga ega. Odatda bu kalit butun son yoki satr shaklida bo‘lib, ma’lumotlar maydonining bir qismini tashkil qiladi. Ro‘yxatlar bilan ishlaganda kalit sifatida ma’lumot maydonlarining turli qismlari ishtirok etishi mumkin. Masalan, familiya, ism, tug‘ilgan yili, ish staji, jinsi kabi ma’lumotlardan tashkil qilingan ro‘yxat familiyalar bo‘yicha tartiblash uchun kalit o‘rnida familiya maydoni, veteranlar izla-nayotgan bo‘lsa ish staji maydonidan foydalanish mumkin.

Ro‘yxatlar ustida quyidagi amallarni bajarish nazarda tutilgan:

- boshlang‘ich ro‘yxat, ya’ni birinchi elementni shakllantirish;
- ro‘yxat oxiriga yangi elementni qo‘shish;
- berilgan kalitli elementni o‘qish ;
- elementni ro‘yxatning ko‘rsatilgan pozitsiyasiga qo‘yish (oldin yoki keyin);
- berilgan kalitli elementni o‘chirish;
- ro‘yxatni kalit bo‘yicha tartiblash.

Ikki tomonlama bog‘langan ro‘yxatni ko‘raylik. Bunday ro‘y-xatni shakllantirish va ishlash uchun hech bo‘lmaganda bitta ko‘r-satkich (ro‘yxatni boshiga ko‘rsatish) lozim bo‘ladi. Ro‘yxatning oxiriga ko‘rsatuvchi yana bitta ko‘rsatkich ishni ancha osonlashtiradi.

Soddalik uchun ro‘yxatni faqat butun sonlardan iborat bo‘lsin, deb faraz qilaylik. Uning elementlari quyidagicha tavsiflanadi:

*struct Node{ int d; Node *next; Node *prev; }:*

Quyida keltirilayotgan dastur 5 ta sondan iborat ro‘yxat tashkil qiladi, yangi element qo‘shadi, elementni o‘chiradi, ro‘yxatni ekranga chiqaradi. Ro‘yxatning boshiga ko‘rsatkich – *pbeg*, oxiriga ko‘rsatkich – *pend*, yordamchi ko‘rsatkichlar – *pv* va *pkey*.

```
#include <iostream.h>
struct Node{
int d;
Node *next;
Node *prev;
};
//---------------
Node * first(int d);
void add(Node **pend, int d);
Node * find(Node * const pbeg, int i);
bool remove(Node **pbeg, Node **pend, int key);
Node * insert(Node * const pbeg, Node **pend, int key, int d);
//-------------
int main()
{
Node *pbeg = first(1);  // 1-elementni shakllantirish
Node *pend = pbeg;     // ro‘yxat boshlanmay turib tugamoqda
// ro‘yxat oxiriga 4 ta element 2, 3, 4 va 5 sonlari qo‘shilmoqda
for (int i=2; i<6; i++)add(&pend, i);
// 2-elementdan keyin 200-element qo‘shilmoqda;
insert(pbeg, &pend, 2, 200);
// 5-element o‘chirilmoqda
if(!remove (&pbeg, &pend, 5)) cout << “Topilmadi”;
Node *pv = pbeg;
while (pv) {       // ro‘yxat ekranga chiqarilmoqda
cout <<pv->d <<’ ‘;
pv = pv->next;
}
 return 0;
}
// ------------------------------------------------
// 1- elementni shakllantirish
Node * first(int d){
```

```
Node *pv = new Node;
pv->d=d;  pv->next=0;    pv->prev = 0;
return pv;
}
// ro‘yxatning oxiriga yangi element qo‘shilmoqda
void add(Node **pend, int d){
Node *pv=new Node;
pv->d=d;  pv->next=0;  pv->prev = *pend;
(*pend)->next = pv;
*pend=pv;
}
// Elementni kalit bo‘yicha izlash
Node * find(Node * const pbeg, int d){
Node *pv = pbeg; while (pv){
if(pv->d == d)break; pv = pv->next;
}
return pv;
}
// Elementni o‘chirish
bool remove(Node **pbeg, Node **pend, int key){
if (Node *pkey = find(*pbeg, key)){     // 1
if (pkey == *pbeg){                    // 2
   *pbeg = (*pbeg)->next;
   (*pbeg)->prev =0;}
  else if (pkey == *pend){          // 3
          *pend = (*pend)->prev;
          (*pend)->next = 0;}
        else{                       // 4
          (pkey->prev)->next = pkey->next;
          (pkey->next)->prev = pkey->prev;}
          delete pkey;
         return true;                  // 5
     }
```

```
    return false;                    // 6
    }
// elementni qo‘shish
Node * insert(Node * const pbeg, Node **pend, int key, int d){
If (Node *pkey = find(pbeg, key)){
    Node *pv = new Node;
     pv->d = d;
   // 1 – yangi tugunni navbatdagi element bilan bog‘lash:
    pv->next = pkey->next;
    // 2 – yangi tugunni oldingi element bilan bog‘lash:
    pv->prev = pkey;
    // 3 – oldingi tugunni yangi element bilan bog‘lash:
    pkey->next = pv;
    // 4 – elementni yangi element bilan bog‘lash:
    if( pkey != *pend) (pv->next)->prev = pv;
    // agar tugun ro‘yxat oxiriga qo‘shilgan bo‘lsa,
    // ro‘yxat oxiriga ko‘rsatkichni yangilash:
     else *pend = pv:
    return pv;
    }
    return 0:
}
```

Ushbu dasturning natijasi quyidagicha:

1 2 200 3 4

Funksiya ichidagi barcha o‘zgarmas parametrlar *const* modifikatori yodramida ko‘rsatilishi lozim. O‘zgaruvchi ko‘rsatkichlar esa (masalan, ro‘yxatning oxirgi elementi o‘chirilganidan keyin qayta tahrirlanishi zarur) adreslar bo‘yicha uzatiladi.

Ro‘yxatdagi elementni o‘chirish funksiyasi *remove* ni batafsil ko‘raylik. Uning parametrlari ro‘yxatning bosh va oxirgi elementlariga ko‘rsatkichlar va o‘chiriladigan element kalitidan iborat. 1- satrda xotiradan lokal *pkey* ko‘rsatkichi uchun joy ajratilmoqda va unga

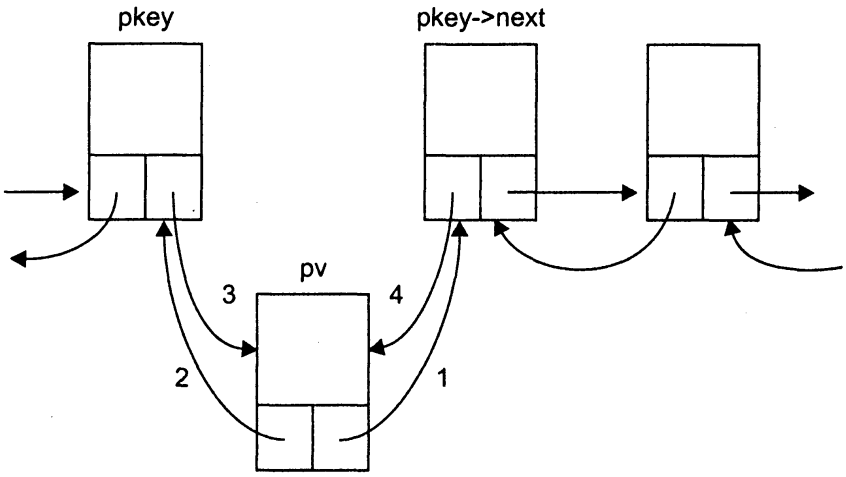
o‘zlashtiriladi. Bu funksiya agar izlangan element topilgan bo‘lsa, ko‘rsatkichni shu elementga keltiradi, aks holda 0 qiymatini oladi. Agar *pkey* nol bo‘lmagan qiymatini olgan bo‘lsa, boshqaruv 2-operatorga uzatiladi, aks holda funksiyadan false qiymati bilan (6- operator) qaytiladi.

Ro‘yxatdan elementlarni o‘chirish ularning qayerda turganligiga bog‘liq ravishda turli xil shaklda bo‘lishi mumkin. 2-operatorda elementning ro‘yxat boshida turganligi tekshirilmoqda. Bu holda *pbeg* ko‘rsatkichini ro‘yxatning navbatdagi elementiga to‘g‘rilash lozim. Bu elementning manzili birinchi elementning *next* maydonida saqlangan. Ro‘yxatning yangi bosh elementining avvalgi elementga ko‘rsatkichi 0 qiymatini olishi lozim.

Agar o‘chiriladigan element ro‘yxat oxirida bo‘lsa, *pend* ko‘rsatkichini manzili oxirgi elementning *prev* maydonida ko‘rsatilgan avvalgi elementga to‘g‘rilanadi. Bundan tashqari, “yangi” oxirgi elementning navbatdagi elementga ko‘rsatkichini nolga aylantirish lozim.

Agar oradagi element o‘chirilayotgan bo‘lsa, avvalgi va keyingi elementlarning ikki tomonlama bog‘lanishlari tashkil qilinadi. Shundan keyin element band qilgan xotira qismi bo‘shatiladi va funksiya *true* qiymatini oladi.

Ro‘yxatga yangi element qo‘shish funksiyasining ishi 14-rasmda keltirilgan. Strelkalarning yo‘nalishlari izohlarda keltirilgan operatorlarga mos keladi.



14-rasm. Ro‘yxatga yangi element qo‘shish.

Ro‘yxatni tartiblash elementlar orasidagi bog‘lanishlarni o‘zgartirishni nazarda tutadi. Bu masala algoritmi berilgan ro‘yxatni ko‘rib chiqish va har bir elementni uning kalitiga mos ravishda yangi ro‘yxatning ko‘rsatilgan pozitsiyasiga qo‘shib qo‘yishdan iborat bo‘ladi.

Quyida tartiblangan ro‘yxatni shakllantirish funksiyasi keltirilmoqda (unda bosh element mavjud deb qaralgan):

```
void add_sort(Node **pbeg, Node **pend. Int d)
{
Node *pv = new Node;    // qo‘shilayotgan element
pv->d = d;
Node * pt = *pbeg;
while (pt){                    // ro‘yxatni ko‘rib chiqish
    if (d < pt->d){            // joriy elementdan oldin kiritish (pt)
    pv->next = pt;
    if (pt == *pbeg){          // ro‘yxat boshiga
pv->prev = 0;
*pbeg = pv;}
else{                          // ro‘yxat o‘rtasiga
    (pt->prev)->next = pv;
    pv->prev = pt->prev;}
    pt->prev = pv;
    return;
    }
    pt = pt->next;
    }
    pv->next=0                 // ro‘yxat oxiriga
    pv->prev= *pend;
    (*pend)->next=pv;
    *pend=pv;
    }
```

11.3. Steklar bilan ishlash

Stek – bu bir tomonlama bog‘langan ro‘yxatning xususiy holi bo‘lib, ma’lumotlarni qarab chiqish faqat uning uchi deb ataladigan tomonidan amalga oshiriladi. Steklar bilan boshqa amallarni bajarish nazarda tutilmagan. Steklar LIFO (last in – first out – oxirgi kelgan birinchi ketadi) prinsipida ishlaydi.

Steklarni sodda qilib ichiga bir nechta to‘p tashlangan tor truba shaklida tasavvur qilish mumkin. Birinchi bo‘lib tashlangan to‘pni olish uchun undan yuqorida turgan barcha to‘plarni olish lozim bo‘ladi. Steklardan tizimli datsuriy ta’minot yoki kompilyatorlar ishlab chiqishda, rekursiv jarayonlarni dasturlashda keng foydalaniladi.

Quyida 5 ta sondan iborat stekni tashkil etib, ularni ekranga chiqaruvchi dastur kodi berilgan. Stekka sonlarni joylash funksiyasi – *push*, tanlash esa *pop*. Stek bilan ishlash funksiyasi – *top* doimo yuqoriga murojaat qiladi.

```
#include <iostream.h>
struct Node
{
int d;
Node * p;
};
Node * first(int d);
void push(Node **top, int d);
int pop(Node **top);
//
int main(){
Node * top=first(1);
for (int I = 2;  i<6;  i++) push (&top, i);
    while (top)
    cout <<pop(&top)<<"   ";
return 0;
}
```

```
// boshlang‘ich stekni shakllantirish
Node * first(int d)
{
 Node *pv = new Node;
 pv->d = d;
 pv->p = 0;
 return pv;
 }
 // stekka kiritish
 void push(Node **top, int d)
 {
 Node *pv = new Node;
 pv->d = d;
 pv->p = *top;
 *top = pv;
}
// stekdan tanlash
int pop(Node **top)
{
int temp = (*top)->d;
Node *pv = *top;
*top = (*top)->p;
delete pv;
return temp;
}
```

Dastur quyidagi natijani beradi:

5 4 3 2 1

11.4. Navbat tashkil qilish

Navbat – bu bir tomonlama bog‘langan ro‘yxatning xususiy holi bo‘lib, yangi elementlar uning bir uchidan, tanlab olish esa ikkinchi uchidan amalga oshiriladi. Navbat uchun boshqa amallar nazarda

tutilmagan. Tanlab olingan elementlar navbatdan chiqariladi. Navbatlar FIFO (first in – first out – birinchi bo‘lib kelgan birinchi bo‘lib ketadi) prinsipi asosida ishlaydi. Dasturlash amaliyotida navbatlar modellashtirish, operatsion tizimni boshqarish masalalarida, buferlashtirilgan kiritish va chiqarishda keng qo‘llanadi.

Quyidagi dastur 5 ta sondan iborat navbatni tashkil qiladi va ularni ekranga chiqaradi. Navbatni oxiriga joylash funksiyasi – *add*, tanlab olish esa *del*. Navbatning boshiga ko‘rsatkich – pbeg, oxiriga ko‘rsatkich – pend.

```
#include <iostream.h>
struct Node
{
  int d;
  Node *p;
};
Node * first(int d);
void add(Node **pend, int d);
int del(Node **pbeg);
//-----
int main(){
Node *pbeg = first(1);
Node *pend = pbeg;
for (int I = 2; i<6; i++)add(&pend, i);
while (pbeg)
cout<<del(&pbeg) <<’ ‘;
return 0;
}
// boshlang‘ich navbatni shakllantirish
Node * first(int d){
Node *pv = new Node;
pv->d = d;
pv->p = 0;
```

```
return pv;
}
// navbatning oxiriga qo‘shish
void add(Node **pend, int d){
Node *pv = new Node;
pv->d = d;
pv->p =0;
(*pend)->p = pv;
*pend = pv;
}
// tanlab olish
int del(Node **pbeg){
int temp = (*pbeg)->d;
Node *pv = *pbeg;
*pbeg = (*pbeg)->p;
delete pv;
return temp;
}
```

Ushbu dastur ishga tushirilsa, quyidagi natijani beradi:

1 2 3 4 5

Takrorlash uchun savol va topshiriqlar

1. Dinamik strukturali ma’lumot deganda nimani tushunasiz?
2. Dinamik strukturali ma’lumotlarning tarkibi qanday bo‘ladi?
3. Ro‘yxat nima va undan qachon foydalanish mumkin?
4. Stek nima uchun qo‘llanadi?
5. Navbat qanday tashkil qilinadi va qaysi prinsip bilan ishlaydi?
6. Quyidagi masalalar uchun dastur ishlab chiqing.

a) Butun sonlardan bir tomonlama bog‘langan ro‘yxat tashkil qiling va uning 4 ga karrali elementlari o‘rta arifmetik qiymatini toping.

b) Butun sonlardan ikki tomonlama bog‘langan ro‘yxat tashkil qiling va uning 5 ga karrali elementlari yig‘indisini toping.

c) Butun sonlardan stek hosil qiling va undagi musbat elementlar koʻpaytmasini toping. Stekdagi maʼlumotlarni koʻrishni tashkil qiling.

d) Haqiqiy sonlar navbatini tashkil qiling va undagi musbat elementlar sonini aniqlang. Navbat maʼlumotlarini koʻrishni tashkil qiling.

12-§. FAYLLAR BILAN ISHLASH TEXNOLOGIYASI

12.1. Umumiy maʼlumotlar

Maʼlumotlarni kompyuter xotira qurilmalaridan birida saqlashning eng qulay shakli fayllar hisoblanadi. Axborotlarni saqlashning boshqa variantlari (masalan, maʼlumotlar bazasi) ham fayllarga asoslanadi.

Fayl — bu kompyuter xotira qurilmalaridan birida saqlanayotgan va oʻz nomiga ega boʻlgan maʼlumotlar toʻplamidir.

Fayl boʻsh boʻlishi ham mumkin. Fayllar maʼlumot saqlashning eng qulay usuli ekanligining sababi quyidagilardan iborat:

1) odatda dasturni bajarib, olingan natijalar dastur oʻz ishini tugatgandan soʻng, EHM xotirasidan oʻchib ketadi. Bu maʼlumotlarga yana ehtiyoj paydo boʻlsa, dasturni yangidan ishga tushirishga toʻgʻri keladi. Buning oldini olish uchun hosil qilingan natijalarni fayllarga yozib qoʻyish mumkin;

2) faylda saqlanayotgan maʼlumotlar koʻplab masala va dasturlar uchun yangi asos boʻlish mumkin, yaʼni dastur natijalari saqlab qoʻyilsa, bu maʼlumotlardan foydalanib boshqa masalalarni yechish mumkin;

3) maʼlumotlar soni EHMning operativ xotirasiga sigʻmaydigan darajada koʻp boʻlishi mumkin. Bunday vaqtda maʼlumotlarning bir qismini biror faylda vaqtincha saqlab qoʻyish mumkin;

4) fayllardan ulardagi maʼlumotlar doirasidagi ixtiyoriy maqsad va masalalar uchun foydalanish mumkin.

Fayllar oʻzining manzili hamda nomiga ega boʻladi. Faylning nomi odatda ikkita qismdan iborat boʻlishi mumkin: nom va kengaytma. Masalan:

D:/CPP/alomat.cpp

yozuvi *alomat.cpp* faylini anglatadi. Bu yerda *alomat* – faylning nomi, *.cpp* esa uning kengaytmasi. Bu faylning manzili – *D* diskdagi *CPP* papkasi.

Dasturchi fayllar ishini tashkil qilar ekan, faqat dastur va uning natijasi haqida qayg‘uribgina qolmasdan, balki ko‘plab qo‘shimcha dasturlar yordamida fayllar yaratish, faylda saqlanayotgan ma’lumotlarni boshqarish, tahlil qilish, tartiblash, ehtiyojga qarab displey yoki qog‘ozda akslantirish kabi masalalarni ham hal qilishi kerak. Yana ilgari ko‘zda tutilmagan yangi ehtiyojlar uchun qo‘shimcha dasturlar yaratish haqida ham o‘ylashi kerak.

C++ tilida fayllar deb, EHMda saqlanayotgan bir xil tipga mansub bo‘lgan ma’lumotlar (komponentalar) to‘plamiga aytiladi.

Faylda saqlanayotgan ma’lumotlardan foydalanish uchun ularni o‘qish va o‘zgaruvchilarga qiymat qilib berish talab qilinadi.

Ixtiyoriy vaqtda faylning faqat bitta komponentasi bilan ishlash mumkin, xolos. Bu ma’lumotni ko‘rsatkich (kursor) ko‘rsatib turadi. Ko‘rsatkich birinchi komponentadan boshlab, har bir ma’lumot o‘qilgandan keyin, navbatdagi o‘qish kerak bo‘lgan ma’lumotni ko‘rsatib turadi. (Boshlang‘ich sinfdagi xatcho‘plarni eslab ko‘ring.)

Fayldagi ma’lumotlar soni o‘zgarib turadi va u dastlab nolga teng bo’ladi. Bu son keyinchalik faylga yangi ma’lumotlar qo‘shilganda ortishi yoki o‘chirilganda nolgacha kamayishi mumkin. Yangi ma’lumotlar doim faylning oxiriga qo‘shiladi.

Dastur yordamida qayta ishlashga mo‘ljallangan fayllar odatda tiplashgan va tiplashmagan bo‘ladi.

Tiplashgan fayllar faqat ma’lum bir tipdagi ma’lumotlarni saqlaydi.

Ma’lum bir tipga mansub bo‘lgan va fayllarda saqlanayotgan ma’lumotlar yozuv deb ataladi. Faylning yozuvlari baytlar bilan o‘lchanadigan chekli hajmga ega va bu hajm barcha yozuvlar uchun bir xil. Har bir yozuvning faylda turgan o‘rnini doimo aniqlash mumkin.

Tiplashmagan fayllar ma’nosi dasturchi tomonidan aniqlanadigan baytlarning chiziqli ketma-ketligini o‘z ichiga oladi.

12.2. Fayllar ustida amallar bajarish

Fayllar ustida qandaydir amallarni bajarish uchun avval ularni ochish lozim. Shundan so‘ng, unda saqlanayotgan ma’lumotlar doirasida quyidagi amallarni bajarish mumkin:

a) faylni yangidan tashkil qilish;

b) yangi ma’lumotlarni yozish (qo‘shish);

c) fayldagi ma’lumotlarni o‘qish.

Bitta fayl bilan joriy vaqtda faqat bitta dastur ishlay oladi holos, sodda qilib aytganda, agar fayl bir dastur tomonidan ochilgan bo‘lsa, u holda boshqa dasturlar shu vaqtning o‘zida bu fayl bilan ishlay olmaydi.

C++ dagi dasturlarda fayllar bilan ishlash quyidagi ketma-ketlikda amalga oshiriladi:

1. Fayl manzilini ko‘rsatgan holda aniq bir maqsad bilan (o‘qish yoki yozish) maxsus funksiya yordamida ochiladi.

2. Fayldagi ma’lumotlarni faylli o‘zgaruvchi yordamida masala algoritmiga mos ravishda qayta ishlash jarayoni tashkil qilinadi.

3. Fayl yopiladi.

Faylli o‘zgaruvchi –faylli tipdagi o‘zgaruvchi bo‘lib, faylning nomi va manzilini bir qiymatli ko‘rsatish uchun xizmat qiladi.

Fayllar bilan ishlash uchun C++ tilida bir qator modullar kutubxonasi mavjud bo‘lib, ularning eng soddasi *fstream.h* sanaladi. Fayllarni qayta ishlash uchun mo‘ljallangan dasturlarda

 # include <fstream.h>

ko‘rsatmasi bo‘lishi shart. Shundan keyin ma’lumot kiritish yoki o‘qish uchun mo‘ljallangan fayllar bilan ishlash imkoniyati paydo bo‘ladi.

Fayllar bilan ishlashning eng oson usuli ma’lumotlar oqimini tashkil qilish bo‘lib, u *Ofstream* xizmatchi so‘zi bilan amalga oshi-

riladi. *Ofstream* dan keyin oqimni nima uchun tashkil qilinayotganligi va manzili ko‘rsatiladi. Ayrim hollarda manzilni tushirib qoldirish ham mumkin. Bu holda fayl joriy katalogdan qidiriladi. Birinchi parametr faylni qanday maqsad uchun ochish kerakligini anglatadi. Maqsadlar quyidagicha bo‘lishi mumkin:

Qiymat	**Ma’nosi**
fin	Faylni o‘qish uchun ochish
fout	Faylni yozish uchun ochish. Agar ko‘rsatilgan nom-dagi fayl mavjud bo‘lsa, u saqlayotgan ma’lumotlar o‘chiriladi va fayl yangidan yaratiladi. Aks holda ko‘rsatilgan manzil va nomdagi fayl hosil qilinadi.

Masalan: *ifstream fin (“D:\CPP\Tub_son.txt”);*
ofstream fout (“Tub_son.txt”).

Faylga yozilishi lozim bo‘lgan ma’lumotlar oqimi

fout << ma’lumotlar ro‘yxati;

buyrug‘i yordamida ko‘rsatiladi. Fayldagi ma’lumotlarni o‘qish masa-lasi esa

fin >> ma’lumotlar ro‘yxati;

ko‘rsatmasi orqali hal qilinadi.

Fayl bilan ishlash tugaganidan keyin uni yopish talab qilinadi. Buning uchun

fout.close(); yoki *fin.close();*

buyruqlaridan foydalanish mumkin.

Odatda fayllar bilan ishlaganda (asosan ma’lumotlar o‘qilganda) ularning uzunligini (mavjud barcha ma’lumotlar hajmini) oldindan bilish mushkul masala. Buning oqibatida yo‘q ma’lumotni o‘qish bilan (ya’ni, faylda ma’lumotlar tugagan) bog‘liq xatolik yuzaga kelishi mumkin.

Faylning tugashi *– shunday vaziyatki, ma’lumotlarning navbat-dagi bo‘lagini o‘qishga uringan kompyuter fayl tugadi (faylning oxiri), degan belgiga duch keladi.*

C++ dasturlash tilida dasturchilarga faylda ma’lumotlar qolgan yoki qolmaganini tekshiruvchi funksiya taklif etiladi:

fin.eof();

Bu funksiya agar faylda o‘qiladigan ma’lumotlar qolmagan bo‘lsa rost (*true*), boshqa hamma hollarda yolg‘on (*false*) qiymatga ega bo‘ladi.

1-masala. 1 dan 1000 gacha bo‘lgan sonlarni *Tub_son.txt* fayliga yozing.

Yechish g‘oyasi. Dastlab *Tub_son.txt* fayli ma’lumotlar oqimini saqlash uchun e’lon qilinadi. So‘ngra 2 dan 1000 gacha bo‘lgan har bir sonning tub yoki tub emasligi tekshiriladi. Buning uchun uni “tub” deb deb faraz qilinadi va qilingan faraz qanday holda o‘rinli bo‘l-masligi, ya’ni uning biror songa qoldiqsiz bo‘linishi tekshiriladi. Agar son tub bo‘lsa, u izlangan ma’lumot sifatida faylga yoziladi.

Bu masalaning dasturi quyidagicha:

```
#include <fstream.h>
# include <math.h>
void main()
{
int i, j, k, t ;
ofstream fout(“tub_son.txt”);      // fayl yozish uchun ochildi
for (i=2; i<=1001; i++)
{
t=1;
k=sqrt(i);
for (j=2; j<=k; j++)
if (i % j == 0) {t=0; j=i+1; }
if (t==1) {fout << i<<” “ ; cout<<i<<”  “;}
}
fout.close();
}
```

Bu dastur ishga tushirilsa, ekranga quyidagi ma’lumotlar chiqariladi:

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997

Aynan ana shu ma’lumotlar joriy katalogdagi “*Tub_son.txt*” fayliga yoziladi. Shundan keyin bu ma’lumotlar doirasida turli amallarni bajarish mumkin bo‘ladi. Jumladan, quyidagi masalani hal qilish mumkin.

2-masala. *Tub_son.txt* faylida 1 dan 1000 gacha bo‘lgan tub sonlar saqlanadi. Ular orasidan egizaklarini toping.

Yechish g‘oyasi. Egizak tub sonlar deb orasidagi farq 2 ga teng bo‘lgan tub sonlarga aytiladi. Shunga ko‘ra, *Tub_son.txt* fayli ma’lumotlar oqimini o‘qish maqsadida ochiladi va undan orasidagi farq 2 ga teng bo‘lganlari ekranga uzatiladi. Ushbu jarayonning dasturi quyidagicha yoziladi:

```
# include<iostream.h>
# include<fstream.h>
void main()
{
int j, k, n;
ifstream fin (“tub_son.txt”);        // fayl o‘qish uchun ochildi
fin >> j  ;
while (!fin.eof())
{
```

```
fin >> k  ;
if (k−j==2) cout <<k<<" va "<<j<<" , ";
j=k;
}
fin.close();
}
```

Ushbu dastur ishga tushirilganda quyidagi ma'lumotlar ekranga uzatiladi:

5 va 3, 7 va 5, 13 va 11, 19 va 17, 31 va 29, 43 va 41, 61 va 59, 73 va 71, 103 va 101, 109 va 107, 139 va 137, 151 va 149, 181 va 179, 193 va 191, 199 va 197, 229 va 227, 241 va 239, 271 va 269, 283 va 281, 313 va 311, 349 va 347, 421 va 419, 433 va 431, 463 va 461, 523 va 521, 571 va 569, 601 va 599, 619 va 617, 643 va 641, 661 va 659, 811 va 809, 823 va 821, 829 va 827, 859 va 857, 883 va 881

Quyidagi dastur joriy katalogdagi *file1.cpp* matnli faylni o'qib, undagi ma'lumotlarni ekranga chiqarish uchun xizmat qiladi:

```
#include "iostream.h"
#include <fstream.h>
void main()
{
// fayldan o'qilgan matnli ma'lumotni
// saqlash uchun o'zgaruvchi kiritamiz
 char buff[80];
ifstream fin("file1.cpp"); // faylni o'qish uchun ochish
while (! Fin.eof()) {     // fayldagi ma'lumotlar tugamagan bo'lsa
fin.getline(buff, 80) ;   // fayldan navbatdagi satr o'qilmoqda
cout<< buff << endl; // bu satr ekranga chiqarilmoqda
}
fin.close();                    // fayl yopilmoqda
return 0;
}
```

Yuqorida ta’kidlanganidek, fayllarni ochish usuli ulardan foydalanish maqsadini belgilab beradi. Bu usullardan birini qo‘llash, ya’ni fayllarni ochish rejimlari uchun ***ios_base*** klassida quyidagi konstantalar nazarda tutilgan.

Fayllarni ochish rejimlari jadvali

Kalit so‘z	**Maqsadi**
ios_base::in	Faylni o‘qish uchun ochish
ios_base::out	Faylni yozish uchun ochish
ios_base::ate	Ochilgan fayl ko‘rsatkichini fayl oxiriga to‘g‘rilash
ios_base::app	Faylni yangi ma’lumotlar bilan to‘ldirish uchun ochish
ios_base::trunc	Agar ko‘rsatilgan fayl mavjud bo‘lsa, undagi ma’lumotlarni o‘chirish
ios_base::binary	Faylni ikkilik sanoq sistemasi rejimida ochish

Fayllarni ochish rejimlarini bevosita obyektni yaratish jarayonida yoki *open()* funksiyasiga murojaat qilayotganda ko‘rsatish mumkin. Masalan, faylni oxiriga yangi ma’lumotlar yozish rejimini o‘rnatish quyidagicha:

ofstream fout(“cppstudio.txt”, ios_base::app);

Fayllarni ochishda bir vaqtning o‘zida bir nechta rejimlardan birgalikda foydalanish ham mumkin. Bunday hollarda “|” belgisi qo‘llanadi. Masalan faylni avval “tozalab”, so‘ngra ma’lumotlarni yozish ko‘rsatmasi quyidagicha yoziladi:

ios_base::out | ios_base::trunc

Ofstream klassi obyektlari to‘g‘ridan-to‘g‘ri

ios_base::out | ios_base::trunc

rejimida ochiladi, ya’ni agar ko‘rsatilgan fayl mavjud bo‘lmasa u yaratiladi, aks holda dastlab unda saqlanayotgan ma’lumotlar o‘chiriladi va bu fayl ma’lumotlarni yozish uchun tayyorlanadi.

Takrorlash uchun savol va topshiriqlar

1. Fayl nima? Uning qanday turlarini bilasiz?
2. Fayllar qanday maqsadlarda ochiladi?
3. Ma’lumotlar oqimini qanday tashkil qilish mumkin?
4. Faylga ma’lumotlar oqimi qanday ko‘rsatma bilan yoziladi?
5. Faylning oxirini aniqlash mumkinmi?
6. Quyidagi masalalar uchun ilova ishlab chiqing:

a) Tub_son.txt faylida 1 dan 1000 gacha bo‘lgan tub sonlar saqlanadi? Har bir yuzlikdagi tub sonlar miqdorini aniqlang.

b) Ruyhat.txt fayliga 20 ta familiya kiriting,

b.1) Ruyhat.txt faylida “Abdullayev” familiyasi mavjudligini aniqlang.

b.2) Ruyhat.txt faylida bir xil familiyalar mavjudligini tekshiring.

b.3) Ruyhat.txt faylida necha xil familiyalar yozilgan?

13-§. GRAFIKLAR BILAN ISHLASH

13.1. Umumiy ma’lumotlar

Displey ekrani gorizontal va vertikal yo‘nalishda zich nuqtalar bilan to‘ldirilgan to‘g‘ri to‘rtburchak shaklidagi maydon bo‘lib, ikki xil rejimda ishlashga mo‘ljallangan:

a) matnli rejim; *b)* grafik rejim.

Matnli rejim faqat dastur matni hamda ma’lumotlarni kiritish uchun mo‘ljallangan bo‘lib, unda ekran ikki xil rang bilan ishlay oladi, xolos. Turbo C muhitining matnli rejimida ekran imkoniyatlari 24x80, ya’ni 24 qatorga ega va har bir qatorga 80 tagacha belgini yozish mumkin.

Grafik rejim esa tasvirlar bilan ishlash, ya’ni turli chizmalar yaratish, tahrir qilish va ko‘rish uchun mo‘ljallangan bo‘lib, matnli rejimdan farqli ravishda ekranning ixtiyoriy nuqtasiga turli rang berish mumkin. Biror rang berilgan nuqtalar to‘plamidan chiziq, belgi va tasvirlarni hosil qilish mumkinligi hammaga ma’lum.

Ekran muayyan vaqtda yuqoridagi rejimlardan faqat birida ishlay oladi, bir vaqtda ikki rejim bilan ishlash imkoniyati hozircha ishlab chiqilmagan. Buning sababi shuki, ekranda kompyuter videoxotira-sidagi ma’lumotlar tasvirlanadi. Joriy vaqtda videoxotirada qanday rejim belgilanganligiga qarab, undagi ma’lumotlar turlicha talqin qilinadi. Videoxotira, nur trubkasi kontrolleri, kiritish va chiqarish qurilmalari displey adapteri deb ataladigan plata tarkibida joylashgan.

Kompyuter imkoniyatlariga ko‘ra bir necha xil displey adap-terlari yaratilgan. Ular ruxsat etilgan imkoniyatlari (eng ko‘p nuqtalar sig‘imi) hamda foydalanish mumkin bo‘lgan ranglarning soni bilan bir-biridan farqlanadi.

Ekranning grafik ish rejimi foydalanilayotgan adapterlarga bog‘liq bo‘lib, sig‘imi 640×480 nuqtali matritsadan boshlab, 1280×1024 yoki undan ham yuqori bo‘lishi mumkin.

Ekran matnli rejimdan grafik rejimga o‘tkazilganidan keyin, C++ tilining grafik imkoniyatlaridan foydalanish mumkin bo‘ladi.

Bunday amal grafik drayverlar deb ataluvchi maxsus dasturlar yordamida bajariladi. Bu drayverlar vazifasini kengaytmasi *.BGI (Borland Graphics Interface*) bo‘lgan fayllar bajaradi.

13.2. Grafik muhitni sozlash

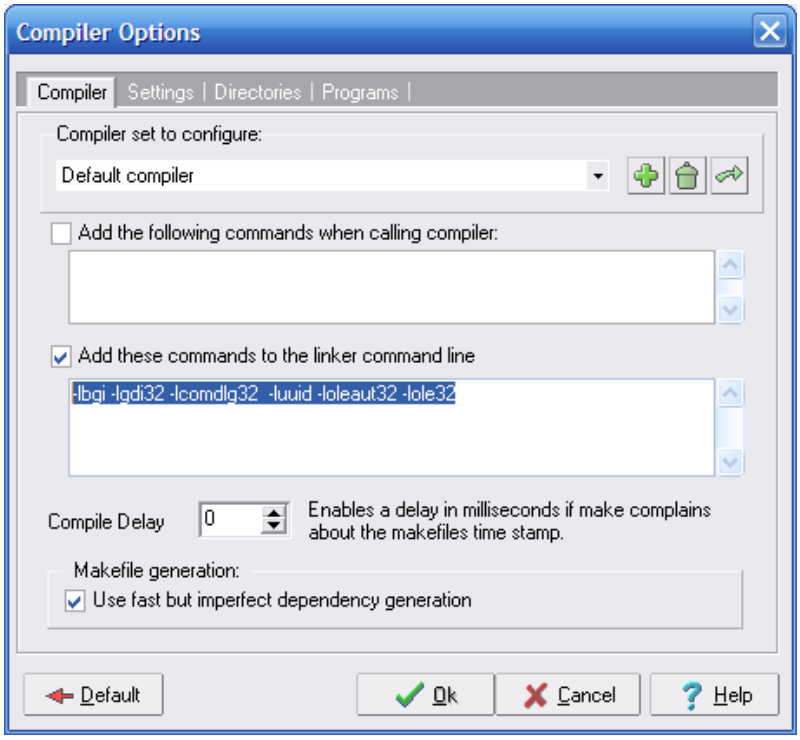
Zamonaviy kompyuterlarda C++ dasturlash tili bilan ishlash uchun Turbo C hamda DEV C++ muhitlari ishlab chiqilgan. Grafiklar bilan ishlash uchun dastlab bu muhitlarni grafik rejimga sozlab olish talab qilinadi.

DEV C++ muhitida grafiklar bilan ishlash uchun *graphics.h* moduli hamda *libbgi.a* kutubxonasi zarur bo‘ladi. Ular ko‘chirib olinganidan so‘ng (bu fayllarni internetdan ko‘chirib olish mumkin), *libbgi.a* faylini DEV C++ o‘rnatilgan papkaning LIB papkasiga, *graphics.h* ni esa INCLUDE papkasiga ko‘chiriladi. So‘ngra DEVCPP.EXE fayli ishga tushiraladi va uning TOOLS oynasidan *Compiler Options* tugmasi bosiladi. Uning ochilgan dialog oynasidagi

Add these commands to the linker command line bayroqchasini o‘rnatib, uning oynasiga

−lbgi −lgdi32 −lcomdlg32 −luuid −loleaut32 −lole32

ko‘rsatmasi qo‘shib qo‘yiladi (15-rasm).



15-rasm. Muhitni grafik rejimga sozlash oynasi.

Eslatma. Grafik bilan bog‘liq bo‘lmagan dastur uchun bu bayroqchani olib qo‘yish talab qilinadi, aks holda dastur ishida xatoliklar yuzaga keladi.

Turbo C dasturlash tilida ishlashni ma’qul ko‘rgan dasturchilar uning *Options* menyusi *Linker* bo’limi *Libraries* punktidagi *Graphics Library* bayroqchasini o‘rnatib qo‘yishlari lozim. Aks holda grafik bilan ishlashning iloji bo‘lmaydi. Grafiklar bilan ishlash uchun qolgan amallar TURBO PASCAL dasturlash tili bilan deyarli bir xil.

13.3. Ekranni grafik holatga o‘tkazish

C++ tilida tasvirlar bilan ishlash uchun *graphics.h* moduli yaratilgan bo‘lib, u o‘z ichiga grafiklar balan ishlash uchun mo’ljallangan ko‘plab vositalarni oladi.

Grafiklar bilan ishlash rejalashtirilgan dasturning bosh qismiga

 # include <graphics.h>

ko‘rsatmasini qo‘shib qo‘yish talab qilinadi.

Bu modulda VGA tipidagi adapterlar uchun drayverlarni ko‘rsatishga mo‘ljallangan 9-konstantasi raqami kiritilgan. Undan tashqari C++ tilida kompyuter uchun eng yaxshi drayverni tanlash operatori *detect* nazarda tutilgan. Undan foydalanish uchun dastur kodiga

int gd=detect;

buyrug‘ini qo‘shib qo‘yish lozim. Shundan keyin ekranni grafik rejimga o‘tkazish uchun ***graphics.h*** modulining ***initgraph*** metodidan foydalanish mumkin bo‘ladi. U umumiy ko‘rinishda quyidagicha yoziladi:

initgraph (gd, gm, “yo‘l”) ;

Bu yerda *gd* – drayver nomeri, *gm* – rejim nomeri, *yo‘l* – tanlangan ish rejimi uchun zarur bo‘lgan drayverlar joylashgan manzilni bildiradi. Agar *gd*=0 bo‘lsa, kompyuter uchun eng yaxshi drayver to‘g‘ridan-to‘g‘ri aniqlanadi.

Yo‘l − kengaytmasi *.BGI* bo‘lgan drayverlar yozilgan katalog manzilidan iborat. Agar drayverlar ko‘rsatilgan manzildan topilmasa, qidirish joriy katalogda davom etadi. Yo‘l umuman ko‘rsatilmagan bo‘lsa, drayverlar joriy katalogdan qidiriladi. Yo‘l noto‘g‘ri ko‘rsatilgan bo‘lsa, zarur drayverni topa olmaganligi sababli kompyuter dasturni umuman bajarmasligi mumkin.

Displeyni grafik rejimdan matnli rejimga qaytarish ham mumkin. Buning uchun *closegraph()* funksiyasidan foydalaniladi.

Quyidagi dasturda grafik ish rejimini ta’minlaydigan va shu zahoti yana displeyning dastlabki ish rejimini qayta tiklaydigan dastur matni keltirilgan.

```
#include <graphics.h>
#include <conio.h>
int main(void)
```

```
{
  int gdriver = detect, gmode;
  initgraph(&gdriver, &gmode, "D:\\TC\\BGI");
  getch();
  closegraph();
  }
```

13.4. Ranglar. Nuqtalar. Chiziqlar

C++ tilida ranglar bilan ishlash uchun maxsus konstantalar hamda xizmatchi so‘zlar kiritilgan.

Rang	T.r.	O‘zbekcha	Rang	T.r.	O‘zbekcha
black	0	qora	darkgray	8	kulrang
blue	1	ko‘k	lightblue	9	moviy
green	2	yashil	lightgreen	10	och yashil
cyan	3	to‘q havorang	lightcyan	11	havorang
red	4	qizil	lightred	12	pushti
magenta	5	binafsha	lightmagenta	13	safsar
brown	6	jigarrang	yellow	14	sariq
lightgray	7	och kulrang	white	15	oq

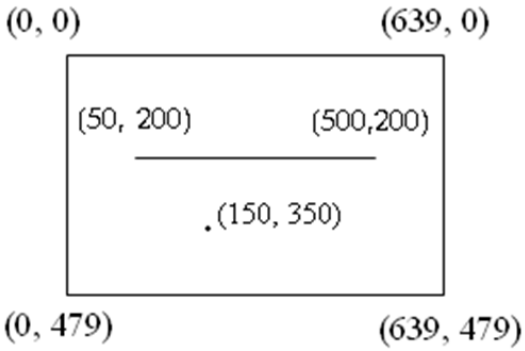
Displeyda hosil qilinadigan geometrik tasvirlarga avval rang tanlash lozim. Bu ish *setcolor* operatori yordamida amalga oshiriladi va umumiy ko‘rinishda

setcolor (rang nomi yoki rangning tartib raqami);

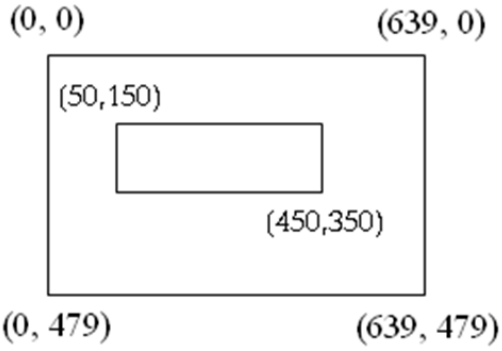
tarzida yoziladi. Masalan,

setcolor(4); yoki *setcolor(red)*;

buyruqlaridan biri bajarilgandan so‘ng, to navbatdagi ***setcolor*** yordamida boshqa rang tanlanmagunga qadar chizmalar qizil rangda hosil qilinadi. Agar rang tanlanmagan bo‘lsa, joriy bo‘lib oq rang xizmat qiladi, ya’ni hamma tasvirlar oq rangda yasaladi.



16-rasm.



17-rasm.

Odatda tasvirlarni chizish vaqtida koordinatalar boshining markazi sifatida qabul qilinadi. Ammo kompyuter ekrani koordinatalari biroz boshqa. 16-rasmda koordinatalarning qay tarzda joylashgani tasvirlangan.

Putpixel metodi displeyda nuqta tasvirini hosil qilish uchun ishlatiladi va umumiy ko‘rinishda

Putpixel (x, y, color);

tarzida yoziladi. Bu yerda *x* va *y* nuqtaning koordinatalari, *color* – rang nomeri yoki nomi. Masalan,

Putpixel (150, 350, red);

ekranning (150, 300) koordinatasini qizil rangda ifodalaydi. (*x, y*) nuqta rang nomerini aniqlash uchun *getpixel(x, y)* funksiyasidan foydalaniladi. Faraz qilaylik, (150, 350) koordinatali nuqtaning rangi qizil bo‘lsin. U holda *getpixel(150, 350)* funksiyaning qiymati 4 ga teng.

Line(x1, y1, x2, y2) – metodi uchlari (*x1, y1*) va (*x2, y2*) nuqtalarda yotgan kesma tasvirini hosil qilish uchun ishlatiladi. Masalan,

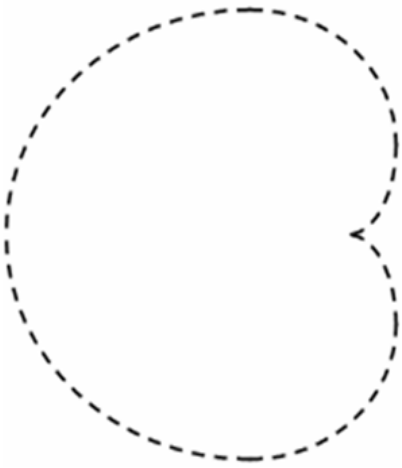
line(50, 200, 500, 200)

buyrug‘i yordamida 16-rasmdagi kesma tasvirini yasash mumkin.

Rectangle(x1, y1, x2, y2) metodidan diagonalining uchlari (*x1, y1*) va (*x2, y2*) nuqtalarda yotgan to‘g‘ri to‘rtburchak tasvirini chizish uchun foydalaniladi.

Rectangle(50, 60, 450, 350);

buyrug‘i natijasi 17-rasmda ko‘rsatilgan.



18-rasm.

Markazlari (*x*, *y*) nuqtada yotgan aylana tasvirini hosil qilish uchun ***circle (x, y, radius)*** metodidan foydalaniladi. Bu yerda *radius* – aylana radiusini bildiradi.

Masala. $x = a\cos t(1+\cos t)$ hamda $y = a\sin t(1+\cos t);\ (t \in [0, 2\pi])$ tenglamalar bilan berilgan kardioida tasvirini yasang.

Yechish g‘oyasi. Bunda *t* ning hamma qiymatlari uchun *x* va *y* o‘zgaruvchilarning qiymatlarini berilgan chiziq tenglamalaridan foydalanib birma-bir hisoblash lozim. Aniqlangan har bir (*x*, *y*) koordinatali nuqta koordinatalar boshi ekranning markaziy qismida (uning koordinatasi (320, 240)) ekanligini hisobga olgan holda, ekranga joylashtiriladi.

Tasvirni biron-bir tugmani bosilgunga qadar ekranda saqlash uchun ***conio.h*** modulining *gath()* metodidan foydalaniladi (18-rasm).

```
#include <graphics.h>
# include <math.h>
#include <conio.h>
int main(void)
{
  int gdriver = detect, gmode, errorcode;
  initgraph(&gdriver, &gmode, "D:\\TC\\BGI");
  float x, y, a=50, fi=0;
  while (fi<=6.28)
  { x=a*cos(fi)*(1+cos(fi))+320;
  y=240-a*sin(fi)*(1+cos(fi));
  fi=fi+0.01;
  putpixel(x, y, 4);
  }
  getch();
  closegraph();
 }
```

13.5. Chizmalarni to‘ldirish (fon berish)

Circle, line va *rectangle* protseduralari figuralarning faqat chetki chiziqlarini chizadi, xolos. C++ tilida ana shu chizmalar egallaydigan sohalarni shtrixovka bilan to‘ldirish (fon berish) imkoniyati yaratilgan. *Setfillstyle* protsedurasi ana shu shirixovkalarni tayinlash uchun xizmat qiladi va

setfillstyle (style, color);

tarzida qo‘llanadi. Bu ko‘rsatmada *style* – joriy shtrixovka nomi yoki kodini, *color* – joriy rang nomi yoki kodini belgilaydi. Shtrixovka nomi va kodlari quyidagicha:

Nomi	Kod	Mazmuni
emptyfill	0	fon rangi bilan to‘ldirish
solidfill	1	berilgan rang bilan to‘ldirish
linefill	2	qalin gorizontal chiziq bilan to‘ldirish
itslashfill	3	ingichka // chiziqlar bilan to‘ldirish
slashfill	4	qalin // chiziqlar bilan to‘ldirish
bkslashfill	5	qalin \\ chiziqlar bilan to‘ldirish
ltbkslashfill	6	ingichka \\ chiziqlar bilan to‘ldirish
hatchfill	7	katakchalar bilan to‘ldirish
xhatchfill	8	qiyshiq katakchalar bilan to‘ldirish
interleavefill	9	qalin // chiziqlari bilan to‘ldirish
widedotfill	10	siyrak nuqtalar bilan to‘ldirish
slosedotfili	11	zich nuqtalar bilan to‘ldirish

Egallaydigan sohasi shtrixovkalar bilan to‘ldiriladigan figuralar uchun quyidagi metodlar kiritilgan:

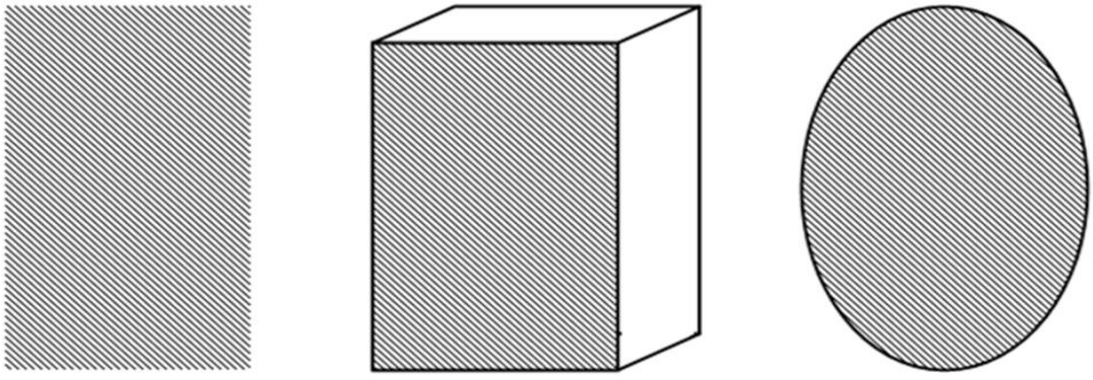
Bar(x1, y1, x2, y2) – diagonali uchlari (*x1, y1*) va (*x2, y2*) nuqtalarda yotgan va joriy shtrixovka bilan to‘ldirilgan to‘g‘ri to‘rtburchakni yasash uchun xizmat qiladi. Bunda uning tomonlari ekran tomonlariga parallel bo‘ladi.

Bar3d(x1, y1, x2, y2, depth, top) – old tomoni joriy shtrixovka bilan to‘ldirilgan parallelepiped tasvirini hosil qilish uchun qo‘llanadi. Bu yerda *depth* – parallelepiped “chuqurligini” ifodalash uchun xizmat qilsa, *top* uning yuqori chiziqlarini chizish yoki chizmaslikni bildiradi. Agap *top* ning qiymati 0 bo‘lmasa, yuqori chiziqlari chiziladi, aks holda yo‘q.

Fillellipse(x, y, xrad, yrad) protsedurasi ichki sohasi joriy shtrixovka bilan to‘ldirilgan ellips chizish uchun ishlatiladi. Ellipsning o‘qlari koordinatalar o‘qiga nisbatan parallel. *Xrad e*llips kengligini, *yrad* uzunligini bildiradi.

Quyidagi dastur natijasi 19-rasmda keltirilgan.

```
#include <graphics.h>
#include <conio.h>
int main(void)
{
 int gdriver = detect, gmode, errorcode;
 initgraph(&gdriver, &gmode, “D:\\TC\\BGI”);
 setfillstyle(11,15) ;
 bar(20,100,100,300) ;
 bar3d(200,100,300,300,40,2) ;
 fillellipse(500,200,20,100);
 getch();
 closegraph();
 }
```



19-rasm.

13.6. Matnlar bilan ishlash

Koʻpincha grafik rejimda ishlab turgan vaqtda, ekranga matnli maʼlumotlarni ham chiqarishga toʻgʻri kelib qoladi. Bu holda avval chiqariladigan matnlar uchun shrift tanlanadi.

C++ tilida ikki xil shrift bilan ishlash imkoniyati mavjud:

a) nuqtali shriftlar; b) vektorli shriftlar.

Nuqtali shrift kattalashtirilganda nuqtalar toʻplamidan iborat ekanligi koʻrinadi. Vektorli shriftda esa bunday emas. Shriftlar kengaytmasi *.ChR* boʻlgan fayllar yordamida aniqlanadi. Shuning uchun biror shriftdan foydalanishga ehtiyoj paydo boʻlganda, dastlab joriy yoki *.BGI* fayllari joylashgan katalogda bu shrift uchun moʻljallangan *.ChR* fayllarining bor-yoʻqligini aniqlab olish lozim.

Shriftlarni tanlash va matnlarni maʼlum bir masshtab bilan chiqarish *settextstyle* protsedurasi yordamida amalga oshiriladi. Uni umumiy holda quyidagicha yozish mumkin:

settextstyle(shrift, yoʻnalish, masshtab);

Bu yerda *shrift* – shrift nomi yoki kodini koʻrsatib, quyidagi son yoki soʻzlardan biri shaklida boʻlishi mumkin:

Shrift	Kodi	Koʻrinishi
defaultfont	0	8x8 standart nuqtali shrift
triplexfont	1	vektorli shrift
smallfont	2	vektorli shrift
sansseriffont	3	vektorli shrift
gothicfont	4	vektorli shrift

Yoʻnalish – matnlarni qanday yoʻnalishda chiqarish kerakligini koʻrsatuvchi oʻzgaruvchi boʻlib, ikki xil qiymat olishi mumkin: *horizdir* – 0 (chapdan oʻngga) va *vertdir* – 1 (pastdan yuqoriga).

Masshtab – belgilarning qanday kattalikda chiqarishni koʻrsatuvchi miqdor hisoblanadi. Normal nuqtali shrift uchun bu miqdor 1 ga, vektor shriftlar uchun esa 4 ga teng.

Ekranga chiqariladigan matn

outtextxy (A, B, ‘matn’);

ko‘rsatmasi yordamida ko‘rsatiladi. Bu yerda *A* va *B* sonlari matn ekranning qaysi nuqtasidan (koordinatalari ko‘rsatiladi) boshlab chiqarilishini belgilab beradi.

Ekranga chiqariladigan matnlarni faqat lotin harflarida yozish shart.

Settextjustufy(horiz, vert) − protsedurasi ekranga chiqariladigan matnni tekislash uchun xizmat qiladi. Bu yerda *horiz* – gorizontal, *vert* esa vertikal tekislashni anglatadi. Tekislash uchun quyidagi nom va kattaliklardan foydalanish mumkin: gorizontal tekislash uchun:

lefttext − 0 chap tomondan tekislash;
centertext − 1 markazga nisbatan tekislash;
righttext − 2 o‘ng tomondan tekislash;
Vertikal tekislash uchun:
bottomtext − 0 past tomondan tekislash;
centertext − 1 markazga nisbatan tekislash;
toptext − 2 yuqoridan tekislash.

Yuqoridagi metodlardan foydalanib, “O‘ZBEKISTON” matnini ikki xil shrift (0 va 1) bilan displeyga chiqarildi.

Quyidagi dastur 20-rasmdagi tasvirni hosil qiladi.

```
#include <graphics.h>
#include <conio.h>
int main(void)
{
 int gdriver = detect, gmode, errorcode;
 initgraph(&gdriver, &gmode, “D:\\TC\\BGI”);
 setfillstyle(11,15) ;
 settextstyle(0,0,5);
 outtextxy(10,100, “O‘ZBEKISTON”);
 settextstyle(1,0,5);
 outtextxy(10,200, “O‘ZBEKISTON”);
 getch();
 closegraph();
 }
```

O‘ZBEKISTON

O‘ZBEKISTON

20-rasm.

13.7. Grafika uchun metod va funksiyalar

Graphics.h moduli hammasi bo‘lib 80 dan ortiq metod va funksiyalarni o‘z ichiga oladi. Ularning hammasini sanab o‘tish anchagina qiyin. Shuning uchun yuqorida aytilmagan va eng ko‘p qo‘llaniladigan funksiya va metodlar hamda ularning vazifalarini keltirib o‘tamiz[3]:

arc(int x, y, bosh_bur, ox_bur, radius) – metodi aylana yoyini chizish uchun xizmat qiladi. Bu yerda *bosh_bur* – yoyning boshlang‘ich burchagi, *ox_bur* – oxirgi burchagi bo‘lib, graduslarda ifodalanadi;

ellipse(int x, y, bosh_bur, ox_bur, Xradius, Yradius) metodi ellips yoyini chizish uchun xizmat qiladi. Bu yerda *bosh_bur* – yoyning boshlang‘ich burchagi, *ox_bur* – oxirgi burchagi bo‘lib, graduslarda ifodalanishi shart;

fillpoly(int bur_soni, int uch_koor) metodi ko‘pburchak tasvirini hosil qilish uchun qo‘llanadi. Bu yerda *bur_soni* – burchaklar soni, *uch_koor* – ko‘pburchak uchlari koordinatalarining massivi;

floodfill(int x, u, ch_r) metodi (*x, y*) nuqta tegishli bo‘lgan sohaga rang berish uchun ishlatiladi. Bu yerda *ch_r* – soha chegarasining rangi;

linerel(int x, y) metodi kursor turgan pozitsiyadagi nuqtani (*x, y*) koordinatali nuqta bilan birlashtiradi;

moveto(int x, y) metodi kursorni (*x, y*) koordinatali nuqtaga o‘tkazish uchun ishlatiladi;

[3] Bu metod va funksiyalarning to‘liq ro‘yxatimi V.V.Podbelskiyning “Язык C++” kitobidan olish mumkin.

pieslice(int x, y, bosh_bur, ox_bur, radius) metodidan sektor tasvirini hosil qilish uchun foydalaniladi. Bu yerda *bosh_bur* va *ox_bur* – mos ravishda sektorning boshlang‘ich va oxirgi burchaklarini gradusda ko‘rsatuvchi o‘zgaruvchi;

outtext(string s) metodi kursor turgan joydan boshlab *s* matnni chiqaradi.

13.8. Harakatli tasvirlar bilan ishlash asoslari

Geometriya kursidan ma’lumki, har qanday figura (tasvir) nuqtalarning qandaydir to‘plamidan iborat. Shuning uchun, ekranda biror tasvir harakatlanganda, uning barcha nuqtalari ham mos ravishda harakatlanayotgan bo‘ladi.

Zamonaviy kompyuter ekranidagi nuqtalarning eng katta sig‘imi (nuqtalar soni) 640×480 yoki undan yuqori bo‘lishi mumkin. Bu sig‘im har bir ekran uchun har xil bo‘lib, operatsion tizimning sozlanishiga bog‘liq bo‘ladi. Ekranda biror tasvirni harakatlantirish uchun uning barcha nuqtalari koordinatalarini ana shu oraliqda bo‘lishini nazorat qilish lozim.

Ekranda birorn-bir figurani harakatlantirish uchun quyidagi ishlarni bajarishga to‘g‘ri keladi:

1) dastlabki pozitsiyada figura tasviri hosil qilinadi;

2) tasvir ma’lum bir vaqt mobaynida ekranda ushlab turiladi;

3) tasvir o‘chiriladi (tasvirga ekran foni rangi bilan bir xil bo‘lgan rang beriladi);

4) tasvirning navbatdagi pozitsiyasi koordinatalari aniqlanadi;

5) tasvir yangi pozitsiyadan hosil qilinadi;

6) ekranning chegaraviy nuqtalarini (sig‘imga mos ravishda) hisobga olgan holda 1–5-bosqich ehtiyojga qarab bajariladi.

1-masala. Aylana bo‘ylab harakat qilayotgan nuqta tasvirini chizing.

Yechish g‘oyasi. Aylana bo‘ylab harakat qilayotgan nuqta tasvirini hosil qilish uchun qutb koordinatalar sistemasidan foydalaniladi.

Ma’lumki, nuqtaning qutb koordinatalari $x = r\cos t$ va $y = r\sin t$ formulalar bilan aniqlanadi. Tasvirni hosil qilgandan so‘ng, uni ma’lum bir muddat ekranda ushlab turish uchun ***dos.h*** modulining ***delay(n)*** metodidan foydalaniladi. Ushbu ma’lumotlar hamda yuqorida sanab o‘tilgan bosqichlarni e’tiborga olgan holda qo‘yilgan masala uchun dasturni quyidagicha yozish mumkin:

```
#include <graphics.h>
#include <conio.h>
# include <math.h>
# include <dos.h>
int main(void)
{
 int gdriver = DETECT, gmode, errorcode;
 initgraph(&gdriver, &gmode, "D:\\TC\\BGI");
 float fi=0, k=50; int x, y;
  do
  {
     fi=fi+0.01;
     if (fi==6.28) fi=0;
     x=k*cos(fi);
     y=k*sin(fi);
     putpixel (x+320, 240-y, 7);
     delay(20);
     putpixel (x+320, 240-y, 0);
     }
     while (fi<6.29) ;
     getch();
 closegraph();
}
```

2-masala. O‘z o‘qi atrofida aylanayotgan kesma tasvirini yasang.

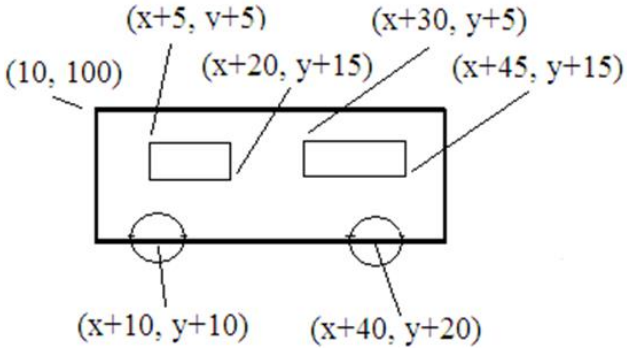
Yechish g‘oyasi. Kesma tasvirini hosil qilish uchun ekranning ikki nuqtasini (qutb koordinatalar sistemasida orasidagi burchagi 180°

bo‘lgan ikkita nuqtani) birlashtirish lozim. Kesmani aylantirish uchun esa uning uchlarini belgilovchi ikki nuqta orqali hosil qilingan kesma tasviridan foydalaniladi. Bu masala uchun yuqoridagi 1-masala asos bo‘lib xizmat qiladi.

```
#include <graphics.h>
#include <conio.h>
# include <math.h>
# include <dos.h>
int main(void)
{
 int gdriver = DETECT, gmode, errorcode;
 initgraph(&gdriver, &gmode, "D:\\TC\\BGI");
 float fi=0, k=50; int x, y, x1, y1;
  do
  {
     fi=fi+0.01;
     if (fi==6.28) fi=0;
     x=k*cos(fi);
     y=k*sin(fi);
     x1=k*cos(fi+3.14);
     y1=k*sin(fi+3.14);
     setcolor(15);
     line(x+320, 240−y, x1+320, 240−y1);
     delay(20);
     setcolor(0);
     line(x+320, 240−y, x1+320, 240−y1);
     }
     while (fi<6.29) ;
     getch();
 closegraph();
}
```

2 – masala. Gorizontal yo‘nalishda harakat qilayotgan avtobus tasvirini hosil qiling.

Yechish g‘oyasi. Bu kabi masalalarni bazaviy nuqtalar usulidan foydalanib hal qilinadi. Bunda tasvirning bitta nuqtasini bazaviy nuqta sifatida qabul qilinadi va qolgan nuqtalar shu nuqta nuqta bilan bog‘lanadi. Natijada bazaviy nuqta o‘z holatini o‘zgartirganda tasvirning qolgan nuqtalari ham unga mos ravishda o‘zgaradi. Soddalik uchun avtobusni bitta to‘rtburchak, ikkita oyna va ikkita g‘ildirakdan iborat deb tasavvur qilaylik (21-rasm). Bazaviy nuqtaning koordinatasi (10, 100). Dastur kodini quyidagicha yozish mumkin:



21-rasm.

```
#include <graphics.h>
#include <conio.h>
# include <math.h>
# include <dos.h>
int main(void)
{
 int gdriver = detect, gmode, errorcode;
 initgraph(&gdriver, &gmode, "D:\\TC\\BGI");
 int x, y;
    x=10; y=100;
    do   {
    setcolor(3);
    rectangle(x, y, x+50, y+20);
    rectangle(x+5, y+5, x+15, y+10);
    rectangle(x+30, y+5, x+45, y+10);
    circle (x+10, y+20, 5);
    circle (x+40, y+20, 5);
```

```
        delay(20);
        setcolor(0);
        rectangle(x, y, x+50, y+20);
        rectangle(x+5, y+5, x+15, y+15);
        rectangle(x+30, y+5, x+45, y+10);
        circle (x+10, y+20, 5);
        circle (x+40, y+20, 5);
        x=x+1;    }
        while (x<=640) ;
        getch();
    closegraph();  }
```

Takrorlash uchun savol va topshiriqlar

1. Kompyuter ekranining qanday ish rejimlari mavjud?
2. Rang tanlash protsedurasini ayting
3. Kesma, to‘g‘ri to‘rtburchak va aylanalar qaysi metod yordamida chiziladi?
4. Grafik rejimda ekranga matnlarni chiqarish mumkinmi?
5. Grafik rejimda matnlarani tekislash usullarini ayting.
6. *Graphics* modulining asosiy protseduralarini sanang.
7. Harakatli tasvirlarni qanday hosil qilish mumkin?
8. Quyidagi masalalar uchun dastur yozing.

a) O‘yinchoq mashina tasvirini hosil qiling.

b) epitsikloida tasvirini chizing.

$$x = (a+b)\cos t - a\cos t((a+b)t/a),$$
$$y = (a+b)\sin t - a\sin t((a+b)t/a),\ b/a > 0,\ t \in [0, 2q\pi]$$

c) O‘zining markaziy nuqtasi atrofida harakatlanayotgan soat millari tasvirini xosil qiling.

d) ekranda mayatnik tasvirini yasang.

e) ekranda yulduzli osmon tasvirini hosil qiling. Unda yulduzlar miltillab tursin.

f) Quyosh, uning atrofida aylanayotgan er tasvirini yarating. Er atrofida oy aylanib tursin.

III BOB. OBYEKTGA ASOSLANGAN DASTURLASH

1-§. OBYEKTGA ASOSLANGAN DASTURLASHGA KIRISH

1.1. Umumiy tushunchalar

XX asrning 90-yillarga kelib masalalarning kattagina sinfini kompyuter yordamida yechish dasturlari ishlab chiqildi. Qolgan masalalar uchun esa imkoniyatlari yanada kengroq bo‘lgan kompyuter va yangi dasturlash tillari zarur bo‘lib qoldi.

Obyektga asoslangan dasturlash texnologiyalari – dasturiy ta’minotning inqiroziga javob sifatida yuzaga kelgan dasturlash texnologiyalari hisoblanadi. Bu inqirozning sababi shunda ediki, strukturali dasturlash metodlari murakkablik darajasi borgan sari ortib borayotgan masalalar uchun dasturiy ta’minot yaratish imkonini bera olmay qoldi. Buning natijasida turli loyihalarni bajarish rejalari buzildi, qilinayotgan xarajatlar belgilangan budjetdan ortib ketdi, dasturiy ta’minot o‘z funksionalligini yo‘qotdi, xatoliklari ortdi.

Dasturiy ta’minotning eng muhim tomonlaridan biri – uning murakkablik darajasidir. Biror dasturchi yolg’iz o’zi qaralayotgan sistemaning barcha xususiyatlarini to‘liq hisobga ola olmaydi. Shu sabab dastur ishlab chiqishda dasturchi va boshqa mutaxassislarning yirik jamoasi qatnashadi. Demak, qo‘yilgan masala bilan bevosita bog‘liq bo‘lgan murakkabliklarga ana shu jamoa ishini yagona maqsad yo‘lida boshqarish ham qo‘shiladi. An’anaviy dasturlash tillarida bunday murakkabliklarni hal qilishda “bo‘lib tashla va hukmronlik qil” tamoyliidan foydalanilgan, ya’ni, masala kichik-kichik masala- larga ajratilib, keyin har bir masala uchun alohida dastur ishlab chiqilgan va birlashtirilgan. Obyektga asoslangan dasturlash texnologiyalari esa masalaga boshqacha usulda yondashadi. Unda masalalarni yechish

uchun kerak bo‘ladigan elementlarni qaralayotgan sohaning turli abstraksiyalariga taalluqli ekanligi asosiy o‘rinda turadi. Bu abstraksiyalar dasturchilar tomonidan ishlab chiqiladi.

Dastlab muayyan bir sohani o‘rganib, uning eng muhim obyektlari va ularga xos bo‘lgan xususiyatlari ajratib olinadi. Ehtiyojga qarab har bir xususiyat ustida bajarish mumkin bo‘lgan amallar aniqlanadi. So‘ngra qaralayotgan sohaning har bir real obyektiga mos dasturiy obyekt o‘ylab topilgan.

Obyektga asoslangan dasturlash texnologiyalarining strukturali dasturlashdan bir qarashdayoq ko‘zga tashlanadigan farqi klasslardan foydalanishdir.

Obyektga asoslangan dasturlash – bu dasturlashning shunday yangi yo‘nalishiki, dasturiy sistema o‘zaro aloqada bo‘lgan obyektlar majmuasi sifatida qaraladi. Har bir obyektni ma’lum bir klassga mansub hamda har bir klass qandaydir shajarani hosil qiladi, deb hisoblanadi. Alohida olingan klass ma’lumotlar to‘plami va ular ustida bajariladigan amallarning majmuasini ifodalaydi. Bu klass elementlariga faqat uning o‘zi uchun aniqlangan amallar orqali murojaat qilish mumkin. Ma’lumotlar va ular ustida bajariladigan amallar o‘rtasidagi o‘zaro bog‘liqlik an’anaviy dasturlash tillariga nisbatan dasturiy sistemalarning ishonchliligini ta’minlaydi. Obyektga asoslangan dasturlashning eng asosiy tushunchasi obyekt va klass hisoblanadi.

Obyekt – bu qaralayotgan buyum yoki hodisaning eng muhim tashkil etuvchilari va ular ustida bajarish mumkin bo‘lgan amallarni yaxlit holda qarash yotadi. Tashkil etuvchilar va ular ustida bajarish mumkin bo‘lgan amallar mos ravishda ***xususiyat*** va ***metodlar*** deb ataladi. Xususiyatni obyektning ***maydoni*** deb ham yuritiladi. Masalan, shashka obyekti rang, vertikal va gorizontal satrlardagi o‘rni kabi maydonlarga, yurish, urish, “damka”ga chiqish kabi metodlarga ega bo‘ladi.

Klass – bu dastur tomonidan aniqlanadigan tip bo‘lib, unda ma’lumotlarning tuzilmasi va Bu tuzilmani qayta ishlashga mo‘ljallangan funksiyalar birlashtiriladi. Bu ma’lumotlar klassning

xususiyatlari, funksiyalar esa maydonlar deb ataladi. Klassning maydonlariga maxsus metodlar yordamida murojaat qilish mumkin. Tuzilishiga koʻra klass obyektga oʻxshab ketsada, unga qaraganda kengroq tushuncha hisoblanadi. Har qanday obyekt muayyan bir klassga mansub boʻladi. Masalan, yuqorida keltirilgan shashka obyekti “Sport oʻyinlari” klassiga tegishli.

Obyektga asoslangan dasturlash asosida ana shu klass va obyektlar oʻrtasidagi oʻzaro aloqani boshqarish algoritmlari yotadi va uch tamoyilga asoslanadi: inkapsulatsiya, vorislik va polimorfizm.

Inkapsulatsiya –maydondagi maʼlumotlarni himoyalash maqsadida koʻzdan yashirish demakdir. Klassning alohida qismlariga maxsus xizmatchi soʻzlar yordamida murojaat qilinadi: public (ochiq qism), private (yopiq qism) va protected (himoyalangan qism).

Ochiq qismdagi maydon va metodlar klass interfeysini tashkil qiladi va ularga erkin murojaat qilish mumkin. Klassning yopiq qismlariga esa faqat klassning xususiy metodlari, himoyalangan qismlarga faqat klassning xususiy metodlari va voris klassning metodlari orqali murojaat qilinadi.

Inkapsulatsiya yordamida maydondagi maʼlumotlarni ehtiyotsizlik natijasida buzib yuborishdan saqlanadi va shu bois dasturning ishonchlilik darajasi ortadi.

Inkapsulatsiya bilan maʼlumotlarni berkitish tushunchasi chambarchas bogʻlangan. Boshqa tomondan, maʼlumotlarni berkitish server va mijoz oʻrtasidagi masʼuliyatni boʻlib olish tushunchasi bilan ham bogʻlangan. Mijoz u yoki bu metodni serverda qanday qilib tashkil qilinganini bilishi shart emas. Mijoz uchun metodning nomi, aniq vazifasi va murojaat qilish usullarini bilish yetarli.

Vorislik deganda mavjud klasslardan foydalanib yangi klasslarni tashkil qilish tushuniladi. Hosila klass oʻz ajdodiga mansub boʻlgan barcha xususiyat va metodlarni voris sifatida qabul qiladi va mavjud klassni oʻzgartirish yoki yangi maʼlumotlarni qoʻshish orqali hosil qilinadi. Masalan, yangi kapalak turi paydo boʻlganda, uni toʻlaligicha yangidan tavsiflash oʻrniga, uning kapalaklar klassiga mansubligi

koʻrsatiladi va boshqa tur kapalaklaridan qaysi jihatlari bilan farq qilishi belgilab beriladi.

Vorislik yordamida klasslarning qarindoshlik shajaralarini ham qurish mumkin.

Polimorfizm – bu turli klasslar tarkibidagi metodlarni bir xil nom bilan atashni anglatadi. Polimorfizm konsepsiyasi obyektga nisbatan metod qoʻllanganida aynan shu obyektning klassiga mos keluvchi metoddan foydalanishni taʼminlaydi. Masalan, shashka obyekti uchun urish va boks obyekti uchun urish metodlari.

1.2. Klasslarni eʼlon qilish

Klasslar C++ tilida obyektga asoslangan dasturlashning eng muhim vositasi sanaladi. Ular strukturaga juda ham oʻxshab ketadi va muayyan obyektga xos boʻlgan maʼlumotlarni ifodalaydi hamda ular ustida bajarish mumkin boʻlgan metodlarni (amallarni) belgilab beradi. Masalan, telefon obyekti uchun tashkil qilingan klass oʻzida raqam, kod va uyali aloqa kompaniyasi kabi maʼlumotlarni saqlaydi va telefon bilan bogʻliq funksiyalarni (dial, answer, hang kabi) ifodalaydi. Klasslar qaralayotgan obyekt haqidagi muhim maʼlumotlarni guruhlaydi va dasturlash jarayonini osonlashtiradi, kodlardan takroran foydalanish imkoniyatini oshiradi.

Obyektga asoslangan dasturlashda dasturchi butun diqqat-eʼtiborini qaralayotgan obyekt-sistemani tashkil etuvchi eng muhim omillar hamda ular ustida bajarish mumkin boʻlgan amallarga qaratadi. Masalan, bu oʻrinda fayl obyekti uchun maʼlumotlarni toʻldirish, tahrirlash, koʻrish, chop etish, yangidan tashkil qilish kabi amallarni aytib oʻtish mumkin. Dasturchi C++ tilida shaxsiy klass va obyektlarini tashkil qilishi va ular yordamida ehtiyojga koʻra lozim topilgan barcha maʼlumotlar va ular ustida bajarish mumkin boʻlgan amallarni tavsiflashi mumkin. Bu klasslardan boshqa dasturlarni ishlab chiqishda ham foydalanish mumkin boʻladi.

Klasslar dasturda nomi, xususiyatlari va metodlarini koʻrsatish orqali aniqlanadi. Bu jarayon maxsus shablon ostida amalga oshiriladi.

Klasslar yordamida dasturda xuddi *int, char* kabi yangi obyektlarni tashkil qilish va qayta ishlash mumkin. Mazkur jarayon umumiy holda quyidagi sxema ostida amalga oshiriladi:

```
class class_nomi
{
   Klass elementlarining (xususiyatlari) tavsifi;
   Metodlar tavsifi;
 };
```

Dastur klass elementlariga qiymatlarni nuqta (“.”) belgisidan foydalanib o‘zlashtiradi. Ana shunday nuqta yordamida klassning metodlariga ham murojaat qilish mumkin.

Klassning nomi betakror, ya’ni boshqa yangi klasslarning nomida uchramasligi lozim. Nomdan keyin figurali qavslar (“{“) ochiladi. So‘ngra bir yoki bir nechta elementlar ro‘yxati ko‘rsatiladi va figurali qavs yopiladi.

Klass aniqlanganidan keyin, unga mansub bo‘lgan obyekt deb ataluvchi o‘zgaruvchilarni e’lon qilish mumkin bo‘ladi. Masalan:

```
class_nomi birinchi_object, ikkinchi_object, uchinchi_object;
```

Quyidagi ko‘rsatmalar bir qator ma’lumot va metodlarga ega bo‘lgan ***xodim*** klassini tavsiflaydi:

```
class xodim
{
  public:
  char nomi[40] ;
  long tab_nom;
  float oylik;
  void show_xodim(void)
  {
    cout << “Ismi va familiyasi:  “ << nomi << endl;
    cout << “Xodimning nomeri: “  << tab_nom<< endl;
    cout << “Maoshi:  “ << oylik << endl;
  };
};
```

Bu misolda qaralayotgan klass uchta xususiyat (nomi, tab_nom, oylik) va bitta metoddan (*show_xodim*) iborat. Koʻrsatmalardagi *public* soʻziga eʼtibor bering. Klassning elementlari xususiy (private) va umumiy (public) boʻlishi mumkin.

Klass aniqlanganidan soʻng, unga mansub boʻlgan obyektlarni (oʻzgaruvchilarni) eʼlon qilish mumkin. Quyidagi namunaga eʼtibor bering.

xodim ishchi, boshliq, kotib; //odam tipidagi oʻzgaruvchilar

Bu koʻrsatma bilan *xodim* klassiga mansub boʻlgan *ishchi, boshliq, kotib* kabi uchta obyekt yaratildi. Demak, bu obyektlar ham *xodim* klassidagi barcha xususiyat va metodlarga ega boʻladi.

Quyidagi dastur *xodim* tipidagi ikkita obyektni yaratadi. Nuqtadan foydalanib, dastur ularga qiymat beradi. Soʻngra *show_xodim* metodi yordamida xodimlar haqidagi maʼlumotlarni chop etadi:

```
#include <iostream.h>
#include <string.h>
class xodim
{
public:
  char nomi [40];
  long tab_nom;
  float oylik;
  void show_xodim(void)
  {
    cout << "Ismi va familiyasi:  " << nomi << endl;
    cout << "Xodimning nomeri:  " << tab_nom<< endl;
    cout << "Maoshi:  " << oylik << endl;
  };
};
void main(void)
{
  xodim ishchi, boshliq;
```

```
   strcpy(ishchi.nomi, “Aliyev Vali”);
   ishchi.tab_nom= 12345;
   ishchi.oylik = 250000;
   strcpy(boshliq.nomi, “Abdullayev Alisher”);
   boshliq.tab_nom= 10001;
   boshliq.oylik = 500000.00;
   ishchi.show_xodim();
   boshliq.show_xodim();
}
```

1.3. Metodlarni klassdan tashqarida aniqlash

Yuqorida keltirilgan misolda *xodim* klassi uchun metod uning ichida (*inline*) e’lon qilingan edi. Bunday metodlarning ko‘payishi dastur ishlab chiqishda bir qator tartibsizlik va chalkashliklarni keltirib chiqarishi mumkin. Buning oldini olish uchun metod prototipini (timsolini) klass ichida e’lon qilish va uni klassdan tashqarida tavsiflash tavsiya etiladi. Quyidagi dasturga e’tibor bering.

```
class xodim
{
public:
   char nomi[40];
   long tab_nom;
   float oylik;
   void show_xodim(void); // funksiya  prototipi
};
```

Turli sinflarda bir xil nomdagi funksiyalardan foydalanish mumkin bo‘lgani uchun tashqarida aniqlanayotgan funksiyalarning nomlari klass nomi va funksiyaga global ruxsat beruvchi operator (::) bilan belgilanishi lozim. Bu amal biz qarayotgan misol uchun quyidagicha yoziladi:

```
void xodim:: show_xodim (void)        //klass nomi
{
cout  << “Ismi va familiyasi:  “ << nomi << endl; // element nomi
cout << “Xodim nomeri:  “ << tab_nom<< endl;
cout << “Maoshi  “ << oylik << endl;
};
```

Ko‘rib turibsizki, keltirilgan kodda funksiya nomi klass nomi (*xodim*) va global ruxsat amali (::) yordamida tavsiflanmoqda. Yakuniy kod quyidagicha yoziladi:

```
#include <iostream.h>
#include <string.h>
class xodim
{
public:
  char nomi [40];
  long tab_nom;
  float oylik;
  void show_xodim(void);
};
void xodim::show_xodim(void)
{
  cout << “Ismi va familiyasi:  “ << nomi << endl;
  cout << “Xodim nomeri: “  << tab_nom<< endl;
  cout << “Maoshi:  “ << oylik << endl;
};
void main(void)
{
  xodim xodim, boshliq;
  strcpy(xodim.nomi, “Aliyev Vali”);
  xodim.tab_nom= 12345;
  xodim.oylik = 250000;
  strcpy(boshliq.nomi, “Abdullayev Alisher”);
```

```
    boshliq.tab_nom= 10001;
    boshliq.oylik = 500000.00;
    xodim.show_xodim();
    boshliq.show_xodim();
  }
```

Takrorlash uchun savol va topshiriqlar

1. Obyektga asoslangan dasturlash nima?
2. Obyektga asoslangan dasturlashning asosiy obyekti nima?
3. Inkapsulatsiya, vorislik va polimorfizm deganda nimani tushunasiz? Javobingizni misollar bilan izohlang.
4. Klasslar qanday e’lon qilinadi va foydalaniladi?
5. Quyidagi masalalarda talab qilingan klasslarni yaratish va foydalanish uchun dastur ishlab chiqing:

a) mullif, nom va bosib chiqarilgan yili kabi ma’lumotlarni o‘z ichiga oluvchi klass;

b) protsessori, operativ xotirasi, vinchester hajmi kabi ma’lumotlardan iborat kompyuter klassi;

c) familiyasi, ismi, mutaxassisligi va kursi kabi ma’lumotlarni o‘z ichiga olgan talaba klassi.

2-§. KONSTRUKTOR VA DESTRUKTOR

2.1. Sodda konstruktor yaratish

Obyektlarni yaratishda eng ko‘p qo‘llanadigan amallardan biri − bu obyekt elementlarini initsializatsiya qilish (boshlang‘ich qiymat berish) sanaladi. Klassning u yoki bu elementiga murojaat qilish uchun uning funksiyalaridan foydalaniladi. Bu jarayonni osonlashtirish uchun C++ tilida konstruktor deb ataladigan shunday maxsus funksiyalar nazarda tutilganki, ular hamma yaratilayotgan klasslar uchun ishga tushishi mumkin. Xuddi shuningdek, C++ tilida obyektlarni o‘chirish (yo‘qotish) uchun destruktor funksiyasi qo‘llanadi.

Konstruktorlar joriy klass elementlarini initsializatsiya qilish metodlari sanaladi va klass bilan bir xil nomga ega bo‘lib, hech qanday qiymat qaytarmaydi.

Destruktorlar ham xuddi konstruktor kabi klass bilan bir xil nomga ega bo‘lib, klasslarning xotiradan egallagan joylarini bo‘shatadi. Ular ham hech qanday qiymat qaytarmaydi.

Yuqorida aytildiki, konstruktor klassning metodlarini ifodalab, klass bilan bir xil nomga ega bo‘ladi. Masalan, *xodim* klassi uchun yaratilgan konstruktorning nomi ham *xodim* bo‘ladi. Agar dasturda konstruktor qurilgan bo‘lsa, yangi obyekt yaratilayotgan paytda C++ uni avtomatik tarzda chaqiradi.

Quyidagi CONSTRUC.CPP dasturi *xodim* nomli klass yaratadi. Shuningdek, dastur obyektga boshlang‘ich qiymat beruvchi *xodim* nomli konstruktorni ham quradi. Bu konstruktor *void* tarzida e’lon qilinmagan bo‘lsa ham hech qanday qiymatni qaytarmaydi.

```
class xodim
{
public:
   xodim(char *, long, float);   // Konstruktor
   void show_xodim(void);
   int change_oylik(float);
   long get_id(void);
private:
   char nomi [64];
   long tab_nom;
   float oylik;
};
```

Konstruktorlarni klassning ixtiyoriy metodi kabi aniqlash mumkin:

```
xodim::xodim(char *nomi, long tab_nom, float oylik)
{
   strcpy(xodim::nomi, nomi) ;
```

```
   xodim::tab_nom = tab_nom;
   if (oylik < 5000000.0)
   xodim::oylik = oylik;
   else   // mumkin bo‘lmagan summa
   xodim::oylik = 0.0;
 }
```

Ko‘rinib turibdiki, konstruktor hech qanday qiymat qaytar-mayapti. Shuningdek, konstruktor global ruxsat beruvchi operatordan foydalanib, har bir elementdan avval klass nomini ko‘rsatmoqda. Quyida CONSTRUC.CPP ning to‘liq matnini keltiramiz:

```
#include <iostream.h>
#include <string.h>
class xodim
 {
   public:
     xodim(char *, long, float);
     void show_xodim(void);
     int change_oylik(float) ;
     long get_id(void);
    private:
      char nomi [64] ;
      long tab_nom;
      float oylik;
   };
   xodim::xodim(char *nomi, long tab_nom,  float oylik)
   {
     strcpy(xodim::nomi, nomi) ;
     xodim::tab_nom = tab_nom;
     if (oylik < 5000000.0)
         xodim::oylik = oylik;
      else  // mumkin bo‘lmagan summa
       xodim::oylik = 0.0;
```

```
}
void xodim::show_xodim(void)
{
 cout << "Xodim        :" << nomi << endl;
 cout << "Tartib nomeri :"  << tab_nom << endl;
 cout << "Oyligi       :" << oylik << endl;
}
void main(void)
{
 xodim ishchi("Aliyev Vali", 101, 200101.0);
 ishchi.show_xodim();
}
```

Ishchi obyektidan keyin xuddi funksiyalardagi kabi oddiy qavslar ichida boshlang‘ich qiymatlar ko‘rsatilganiga e’tibor bering.

Obyekt e’lon qilinayotganda ularning parametrlarini konstruktorga uzatish mumkin. Bu ko‘rsatma umumiy ko‘rinishda

class_name object(valuel, value2, value3)

tarzida yoziladi. Masalan, agar dasturda bir nechta *xodim* tipidagi obyektlar e’lon qilinsa, ularning har bir elementini konstruktor yordamida quyidagicha initsializatsiya qilish mumkin:

```
xodim ishchi("Alijonov Valijon      ", 101, 200101.0);
xodim secretary("Daminov Sodiqjon", 57,  150000.0);
xodim manager("Bahodirov Botir     ", 1022, 300000.0);
```

Ma’lumki, funksiya parametrlari ko‘rsatilmasa, C++ qiymatlarni to‘g‘ridan-to‘g‘ri yozishga imkon beradi, ya’ni funksiya parametrlari ko‘rsatilmagan dasturning ishlashi vaqtida, bu qiymatlar to‘g‘ridan-to‘g‘ri tayinlanadi. Konstruktor ham bundan istisno emas. Dasturda xuddi funksiyalardagi kabi bu qiymatlarni to‘g‘ridan-to‘g‘ri ko‘rsatish mumkin. Masalan, quyidagi kodda *xodim* konstruktori oylik maoshni to‘g‘ridan-to‘g‘ri 100000 qilib belgilaydi.

```
Xodim::xodim(char *nomi, long tab_nom,
 float oylik =100000.00)
 {   strcpy(xodim::nomi, nomi);
     xodim::tab_nom = tab_nom;
     if (oylik < 5000000.0)
     xodim::oylik = oylik;
     else  // mumkin bo‘lmagan summa
     xodim::oylik = 0.0;  }
```

2.2. Konstruktorlarni qayta yuklash

C++ tili funksiyalarni boshqa tipdagi parametrlar uchun qayta yuklash imkonini beradi. Shuningdek, konstruktorlarni ham qayta yuklash mumkin. Quyidagi dastur *xodim* konstruktorini qayta yuklaydi. Birinchi konstruktor xodimning ismi, tartib nomeri va maoshini ko‘rsatishni talab qiladi. Ikkinchi konstruktor esa foydalanuvchidan bu ma’lumotlarni kiritishni so‘raydi:

```
xodim::xodim(char *nomi, long tab_nom)
{
  strcpy(xodim::nomi, nomi);
  xodim::tab_nom = tab_nom;
  do
 { cout <<nomi<<“ uchun 5000000 dan kam bo‘lgan maoshini
 kiriting:“;
  cin >> xodim::oylik;   }
  while (oylik >= 5000000.0);
}
```

Klassni aniqlashda har ikki konstruktor uchun timsollarni kiritish lozim:

```
class xodim
{
public:
```

```
xodim (char *, long, float); // qayta yuklash timsollari
xodim(char *, long);
   void show_xodim(void);
   int change_oylik(float);
   long get_id(void);
 private:
   char nomi [64];
   long tab_nom;
   float oylik;
 }
```

Quyida ana shu dasturning to‘la matnini keltiramiz:

```
 #include <iostream.h>
 #include <string.h>
 class xodim
 {
 public:
   xodim(char *, long, float);
   xodim(char *, long);
   void show_xodim(void);
   int change_oylik(float) ;
   long get_id(void);
 private:
   char nomi [64];
   long tab_nom;
   float oylik;
 };
 xodim::xodim(char *nomi, long tab_nom,
float oylik)
 {
   strcpy(xodim::nomi, nomi);
   xodim::tab_nom = tab_nom;
   if (oylik < 5000000.0) xodim::oylik = oylik;
```

```
   else  // Mumkin bo‘lmagan summa
      xodim::oylik = 0.0;
 }
 xodim::xodim(char *nomi, long tab_nom)
 {
   strcpy(xodim::nomi, nomi);
   xodim::tab_nom = tab_nom;
   do
   {
cout <<nomi<< “ uchun 5000000 dan kam bo‘lgan maoshini
kiriting: “;
   cin >> xodim::oylik;
   }
   while (oylik >= 5000000.0);
 }
 void xodim::show_xodim(void)
 {
cout << “Xodim            :” << nomi << endl;
cout << “Tartib nomeri :”  << tab_nom << endl;
cout << “Oyligi           :” << oylik << endl;
 }
 void main(void)
 {
xodim ishchi(“Aliyev Vali”, 101, 200101.0);
xodim manager(“Bahodirov Botir     ”, 1022, 300000.0);
ishchi.show_xodim();
manager.show_xodim();
 }
```

Dasturni kompilatsiya qilib, ishga tushirilganda u Bahodirov Botir uchun 5000000 dan kam bo‘lgan maosh summasini kiritishni so‘raydi. Ma’lumot kiritilganidan so‘ng, har ikki xodim haqidagi ma’lumotlar ekranga chiqariladi.

2.3. Destruktor haqidagi boshlang‘ich ma’lumotlar

Destruktorlar dastur yordamida yaratilgan obyektlarni yo‘q qilish vaqtida to‘g‘ridan-to‘g‘ri ishga tushadi. Dinamik obyektlarni saqlash uchun dastur kompyuter xotirasidan joy ajratadi, so‘ngra bajarish vaqtida yaratilgan obyektlarni yo‘q qiladi. Bunday jarayonlarni dasturlashda destruktorlardan foydalanish maqsadga muvofiq bo‘ladi.

Yuqoridagi misollarda obyektlar dasturni bajarish vaqtida hosil qilingan edi. Dastur o‘z ishini tugatishi bilan C++ bu obyektlarni yo‘q qilgan. Agar destruktorlar dastur ichida aniqlansa, C++ dastur ishini tugatmay turib bu destruktorlarga to‘g‘ridan-to‘g‘ri murojaat qiladi.

Konstruktor kabi destruktorlar ham klass bilan bir xil nomga ega bo‘ladi. Ammo destruktorda bu nomdan avval tilda belgisi (~) yozilishi lozim. Bu ko‘rsatma umumiy ko‘rinishda quyidagicha yoziladi:

```
~class_nomi (void)  //destruktorga ko‘rsatkich
{
// destruktor operatorlari
}
```

Konstruktorlardan farqli ravishda destruktorlarga parametrlarni uzatib bo‘lmaydi. Quyidagi dastur *xodim* klassi uchun destruktor e’lon qiladi:

```
void xodim:: ~xodim(void)
{
  cout << nomi <<” uchun ob’yekt yo‘q qilindi “ << endl;
}
```

Bu misolda destruktor ekranga C++ obyektni yo‘q qilayotganligi haqidagi axborotni chiqaradi. Dastur o‘z ishini tugatganidan keyin, C++ har bir obyekt uchun destruktorlarni to‘g‘ridan-to‘g‘ri chaqiradi. Quyida shu dasturning to‘liq kodi keltirilgan:

```
#include <iostream.h>
#include <string.h>
class xodim
{
public:
   xodim(char *, long, float);
   ~xodim(void);
   void show_xodim(void);
   int change_oylik(float);
   long get_id(void);
private:
   char nomi [64] ;
   long tab_nom;
   float oylik;
};
xodim::xodim(char *nomi, long tab_nom, float oylik)
{
   strcpy(xodim::nomi, nomi) ;
   xodim::tab_nom = tab_nom;
   if (oylik < 5000000.0)  xodim::oylik = oylik;
   else  // mumkin bo‘lmagan summa
   xodim::oylik v 0.0;
}
void xodim:: ~xodim(void)
{
   cout <<nomi<< “ uchun obyekt yo‘q qilindi “ << endl;
}
void xodim::show_xodim(void)
{
   cout << “Xodim ismi     : “ << nomi << endl;
   cout << “Tartib nomeri: “  << tab_nom << endl;
   cout << “Oyligi            : “ << oylik << endl;
}
```

```
 void main(void)
 {
xodim ishchi(“Aliyev Vali”, 101, 200101.0);
ishchi.show_xodim();
 }
```

Ushbu dastur kompilatsiya qilinganidan so‘ng ishga tushirilganda, ekranda quyidagi axborot chiqariladi:

```
Xodim ismi    : Aliyev Vali
Tartib nomeri : 101
Oyligi        : 200101
Aliyev Vali uchun obyekt yo‘q qilindi
```

Ko‘rinib turibdiki, dastur to‘g‘ridan-to‘g‘ri destruktorni chaqirmoqda.

Hozirgacha qaralgan dasturlarda destruktorlardan foydalanishga ortiqcha ehtiyoj bo‘lmagan edi. Ammo dasturda ko‘plab obyektlarni e’lon qilishga to‘g‘ri kelganda, kompyuter xotirasidan unumli foydalanish uchun bu obyektlarning xotiradan band qilgan joylarini bo‘shatish uchun destruktorlar qulay vosita hisoblanadi.

Takrorlash uchun savol va topshiriqlar

1. Konstruktor nima va u qanday vazifani bajaradi?
2. Konstruktorlarda qiymat ko‘rsatilmasa nima bo‘ladi?
3. Destruktor nima?
4. Destruktorlardan foydalanishning ahamiyati nimada?
5. Quyidagi masalalarni hal qilishda konstruktor va destruktorlardan foydalaning.

a) Talabalarning familiyasi, ismi va informatika fanidan olgan baholarini faylga yozing. Eng ko‘p ball to‘plagan talabani aniqlang.

b) Avtomobillarning markasi, dvigatel quvvati, nomerlarini faylga kiriting. Eng katta quvvatga ega bo‘lgan avtomobilni aniqlang;

c) Mahsulotning ishlab chiqarilgan yili, yaroqlilik muddati va miqdori berilgan bo‘lsin. Joriy yilda yaroqlilik muddati tugagan mahsulotlarning umumiy miqdorini aniqlang.

3-§. VORISLIK

3.1. Umumiy ma’lumotlar

Vorislik obyektga asoslangan dasturlashning fundamental konsepsiyasi hisoblanib, mavjud obyektlardan (klasslardan) yangilarini yaratishda foydalanish imkoniyatiga ishora qiladi. Faraz qilaylik, qandaydir klass mavjud hamda asosiy parametrlari ana shu klassnikiga o‘xshagan, ammo undan ma’lum bir parametr yoki metodlari bilan farqlanadigan boshqa klassni qurish talab qilingan bo‘lsin. Bunday hollarda C++ dasturlash tili talab qilingan yangi klassni (obyektni) ana shu mavjud klass orqali tavsiflash imkonini beradi. Hayotdan shunday misol keltirish mumkin. Olimlar kapalakning yangi turini tutib olishdi. Uni ta’riflashda fanga oldindan ma’lum bo‘lgan kapalaklarga xos umumiy belgilardan foydalaniladi va qo‘shimcha ravishda yangi kapalak ulardan nimasi bilan farq qilishi ko‘rsatiladi.

Obyektlarni bu usulda tavsiflashda yangi obyektni hosila, uni ta’riflashda foydalanilgan bazaviy obyekt avlod deb ataladi. Hosila obyekt avlodiga xos bo‘lgan barcha parametr va metodlarni o‘ziga oladi, ya’ni vorislik qiladi. Sodda qilib aytganda, bazaviy klassning asosiy parametrlari hosila klassiga o‘tadi.

Hosila klassi elementlarini initsializatsiya qilish uchun bazaviy va hosila klass konstruktorlaridan foydalaniladi.

Nuqta operatorini qo‘llab hosila va bazaviy klass elementlariga murojaat qilish mumkin.

Umumiy (*public* – dasturning barcha qismida foydalanish mumkin bo‘lgan) hamda xususiy (*private* – faqat klass metodlari uchun mumkin bo‘lgan) elementlar bilan bir qatorda C++ himoyalangan (*protected*) elementlar bilan ham ishlash imkoniyatlarini taqdim etadi.

Bazaviy va hosila klasslar ma’lumotlariga murojaat qilinganda anglashilmovchiliklar yuzaga kelishi mumkin. Chunki vorislik mexanizmiga ko‘ra ulardagi parametrlar uchun tashkil qilingan maydonlar

bir xil nomga ega. Bu muammoni bartaraf etish uchun dastur global ruxsat beruvchi amal (::) operatoridan bazaviy yoki hosila klassi nomini ko‘rsatib foydalanishi mumkin.

3.2. Sodda vorislik

Yuqorida mulohazalarga ko‘ra, vorislik hosila klassining bazaviy klassga xos alomatlarni o‘ziga “meros” qilib olish qobiliyati bilan belgilanadi. Faraz qilaylik, bizda bazaviy *xodim* klassi mavjud bo‘lsin:

```
class xodim
{
public:
xodim(char *, char *, float);
void show_xodim(void);
private:
char nomi[64];
char vazifa[64];
float oylik;
};
```

Dasturda o‘ziga *xodim* ga qo‘shimcha ravishda quyidagi elementlarni oluvchi *manager* klassini qurish talab qilingan bo‘lsin:

```
float annual_bonus;
char company_car[64];
int stock_options;
```

Bu misol uchun dasturchi ikkita variantdan birini tanlashi mumkin:

1) mutlaqo yangi *manager* klassini tashkil qilib, *xodim* klassiga xos bo‘lgan ko‘plab parametrlarni takroran yozishga to‘g‘ri keladi;

2) *manager* klassi *xodim* asosida tashkil qilinadi.

Ikkinchi variant qo‘llanganda dastur hajmi sezilarli darajada kamayadi va kodlarni takroran yozishning oldi olinadi.

Voris klassini e’lon qilish uchun *class* kalit so‘zi, so‘ngra hosila klass nomi, ikki nuqta belgisi hamda bazaviy klass nomi ko‘rsatilishi lozim. Yuqoridagi masala uchun bu ko‘rsatma quyidagicha yoziladi:

```
class manager  :  public xodim {  // Bazaviy klass
 // bu yerda elementlar aniqlanadi
};
```

Xodim klassi nomidan avval ko‘rsatilgan *public* kalit so‘zi bu klassning umumiy (*public*) elementlari *manager* klassi uchun ham umumiy ekanligini anglatadi. Quyidagi operatorlar *manager* klassini hosil qiladi:

```
 class manager : public xodim
 {
public:
manager(char *, char *, char *, float, float, int);
void show_manager(void);
private:
float annual_bonus;
char company_car[64];
int stock_options;
};
```

Bazaviy klassdan yangi klass hosil qilinganda, hosila klassining xususiy elementlari bilan ishlash faqat bazaviy klassning interfeys funksiyalari orqali mumkin bo‘ladi, xolos. Boshqacha aytganda, hosila klass bazaviy klass elementlariga nuqta operatori yordamida to‘g‘ri-dan-to‘g‘ri murojaat qila olmaydi.

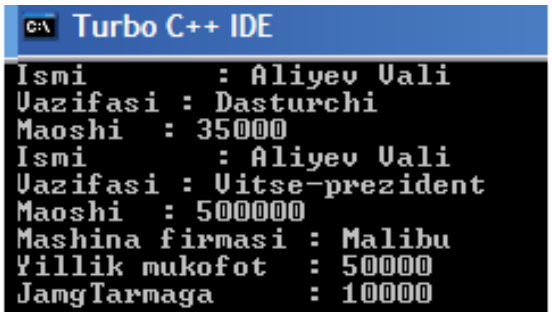
Quyidagi dastur *xodim* klassi asosida yangi *manager* klassini qurish jarayonini namoyish qiladi:

```
#include <iostream.h>
 #include <string.h>
 class xodim
 {
```

```
  public:
  xodim(char *, char *, float);
  void show_xodim(void);
  private:
  char nomi [ 64 ];
  char vazifa[64];
  float oylik;
  };
 xodim::xodim(char *nomi, char *vazifa,float oylik)
 {
  strcpy(xodim::nomi, nomi);
  strcpy(xodim::vazifa, vazifa);
  xodim::oylik = oylik;
}
void xodim::show_xodim(void)
{
  cout << "Ismi      : " << nomi << endl;
  cout << "Vazifasi : " << vazifa << endl;
  cout << "Maoshi  : " << oylik << endl;
 }
 class manager : public xodim
  {
  public:
  manager(char *, char *, char *, float, float, int);
  void show_manager(void);
  private:
  float bonus;
  char marka[64];
  int fond;
  };
  manager::manager(char *nomi, char *vazifa, char *marka,
float oylik, float bonus, int fond) :  xodim(nomi, vazifa, oylik)
 {
```

```
  strcpy(manager::marka, marka) ;
  manager::bonus = bonus ;
  manager::fond = fond;
}
void manager::show_manager(void)
{
  show_xodim();
  cout << "Mashina firmasi  : " << marka << endl;
  cout << "Yillik mukofot     : " << bonus << endl;
  cout << "Jamg'armaga     : " << fond << endl;
  }
 void main(void)
{
xodim ishchi("Aliyev Vali", "Dasturchi", 35000);
manager boss("Aliyev Vali", "Vitse-prezident", "Malibu",
500000.0, 50000, 10000);
ishchi.show_xodim() ;
  boss.show_manager();
  }
```

Ushbu dastur quyidagi natijani beradi:



Ko'rinib turibdiki, dastur avval bazaviy *xodim* klassi, so'ngra uning asosida voris *manager* klassini yaratmoqda. Hosila klassini yaratishda uning konstruktori bazaviy klass konstruktoriga murojaat qilishi lozim. Buning uchun ikki nuqta belgisini hosila klassi konstruktoridan keyin yozib, bazaviy klass konstruktorini zarur parametrlar bilan birgalikda ko'rsatish talab qilinadi:

```
manager::manager(char *nomi, char *vazifa, char *marka,
float oylik, float bonus, int fond) :
xodim(nomi, vazifa, oylik)  // bazaviy klass konstruktori
{
strcpy(manager::marka, marka);
manager::bonus = bonus;
manager::fond = fond;
}
```

Shuningdek, *show_manager* funksiyasi *show_xodim* ga murojaat qilmoqda. *Manager* klassi *xodim* ning hosila klassi bo‘lgani uchun uning umumiy elementlariga xuddi o‘z elementlariga kabi murojaat qilishi mumkin.

Faraz qilaylik, mavjud dastur tarkibida quyidagi *book* klassidan foydalanilayotgan bo‘lsin.

```
Class book
{
public:
book (char *, char *, int);
void show_book(void);
private:
char titul[64];
char author[b 4];
int pages;
};
```

Bu klass asosida *library_card* klassini yaratish talab qilingan bo‘lsin. Unga quyidagi qo‘shimcha parametrlarni kiritish lozim:

```
char catalog[64];
int checked_out; // 1, agar tekshirilgan bo‘lsa, aks holda − 0
```

book klassidan *library _card* klassini hosil qilish quyidagicha amalga oshiriladi.

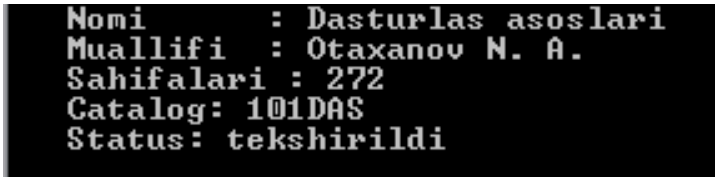
```
Class library_card : public book
{
  public:
     library_card(char *, char *, int, char *, int);
     void show_card(void);
  private:
       char catalog[64] ;
       int checked_out;
 };
```

Dasturning to‘la matnini keltiramiz:

```
#include <iostream.h>
#include <string.h>
class book
{
public:
    book(char *, char *, int);
    void show_book(void);
 private:
    char title [64];
    char author[64];
    int pages;
 };
    book::book(char *title, char *author, int pages)
    {
       strcpy(book::title, title);
       strcpy(book::author, author);
       book::pages = pages;
     }
     void book::show_book(void)
     {
        cout << “Nomi      : “ << title << endl;
        cout << “Muallifi : “ << author << endl;
```

```
    cout << "Sahifalari : " << pages << endl;
  }
  class library_card : public book
 {
  public:
    library_card(char *, char *, int, char *, int);
    void show_card(void) ;
 private:
    char catalog[64];
    int checked_out;
  };
library_card::library_card(char *title, char *author, int pages,
char *catalog, int checked_out) :  book(title, author, pages)
{
  strcpy(library_card::catalog, catalog) ;
  library_card::checked_out = checked_out;
}
void 1ibrary_card::show_card(void)
{
  show_book() ;
  cout << "Katalog: " << catalog << endl;
  if (checked_out)  cout << "Status: tekshirildi" << endl;
  else  cout << "Status: erkin" << endl;
}
void main(void)
{
 library_card card( "Dasturlash asoslari", "Otaxanov N.A.",
272, "101DAS", 1);
 card.show_card();
}
```

Ushbu dastur ishga tushirilganda ekranda quyidagi ma’lumotlar paydo bo‘ladi:



Avvalgi misoldagi kabi, *library_card* konstruktori *book* klassi konstruktoriga murojaat qilmoqda. Bundan tashqari, *book* klassining *show_card* funksiyasi ichida qo‘llanishiga ham e’tibor bering. *Library_card* klassi *book* klassining metodlarini o‘ziga olgani uchun *show_card* metodi *show_book* ni nuqta operatorisiz ham chaqirishi mumkin.

3.3. Himoyalangan elementlar

Bazaviy klass elementlarini aniqlashda ularni *public, private* yoki *protected* (umumiy, xususiy yoki himoyalangan) tarzida e’lon qilinganini ko‘rdik. Hosila klassi bazaviy klass elementlaridan xuddi o‘ziniki kabi foydalanishi mumkin. Ammo hosila klassi bazaviy klassning xususiy elementlariga to‘g‘ridan-to‘g‘ri murojaat qila olmaydi.

Himoyalangan elementlar xususiy va umumiy elementlar o‘rtasidagi oraliq vaziyatni egallaydi. Agar element himoyalangan bo‘lsa, hosila klassi elementlari ularga xuddi umumiy elementlar kabi murojaat qilishi mumkin. Dasturning qolgan qismi uchun himoyalangan elementlar xususiyga aylanadi. Himoyalangan elementlardan foydalanishning yagona usuli interfeysli funksiyalar yordamida amalga oshiriladi. *Book* klassini tavsiflashda *protected* tamg‘asidan quyidagicha foydalanish mumkin:

```
class book
{
public:
  book(char *, char *, int) ;
  void show_book(void) ;
protected:
  char title [64];
  char author[64];
  int pages;
};
```

Bu holat *book* ning voris klasslari uchun *title*, *author* va *pages* elementlarga nuqta operatori bilan to‘g‘ridan-to‘g‘ri murojaat qilish imkonini beradi.

Agar yaratilayotgan klasslardan kelajakda yangi klasslarni hosil qilish uchun foydalanish rejalashtirilgan bo‘lsa, hosila klasslarining elementlari yangi yaratilayotgan klass elementlariga to‘g‘ridan-to‘g‘ri murojaat qilishi mumkinligini nazarda tutish va bunday elementlarni himoyalangan holda ishlab chiqishni tashkil qilish lozim.

Himoyalangan elementlar o‘zlari bilan ishlash hamda himoya-lash imkoniyatlarini ta’minlab beradi.

Yuqoridagi mulohazalardan ko‘rish mumkinki, voris klasslar bazaviy klass elementlariga nuqta operatori orqali murojaat qila olganda vorislik mexanizmi yaxshi samara beradi. Bunday hollarda dastur klasslarning himoyalangan elementlaridan foydalana oladi va hosila klass bazaviy klass elementlariga nuqta operatori yordamida to‘g‘ridan-to‘g‘ri murojaat qilishi mumkin bo‘ladi.

Bir klass ikkinchisidan hosil qilinganda, hosila klassidagi elementlarning nomi bazaviy klass elementlari nomi bilan bir xil bo‘lib qolishi mumkin. Bunday vaziyatlarda C++ tili doimo hosila klassi ichida uning elementlaridan foydalanadi. Masalan, aytaylik, *book* va *library_card* klasslari *price* elementidan foydalanilgan bo‘lsin. *Book* klassi misolida *price* elementi kitobning narxiga (masalan, 10000 so‘m) mos keladi. *Library_card* klassida esa *price* elementi kitobning chegirma narxini (masalan, 8750 so‘m) anglatishi mumkin.

Agar oshkor holda (:: global ruxsat operatori yordamida) ko‘rsatilmagan bo‘lsa, *library_card* klassi funksiyalari hosila klassi (*library_card*) elementlaridan foydalanadi. *Library_card* klassi funk-siyalari bazaviy *book* klassining *price* elementiga murojaat qilish uchun *book* klassi nomini va ruxsat berish operatorini ko‘rsatish lozim. Masalan,

book::price;

Faraz qilaylik, *show_card* metodi har ikki narxni ekranga chiqarishi talab qilingan bo‘lsin. Bu buyruqni quyidagicha yozish mumkin:

```
cout << “Haqiqiy bahosi: so‘m” << price << endl;
cout << “Sotuvdagi narxi: so‘m” << book::price << endl;
```

Takrorlash uchun savol va topshiriqlar

1. Obyektga asoslangan dasturlash deganda nimani tushunasiz?
2. Vorislik mexanizmini izohlab bering.
3. Vorislik mexanizmini qo‘llashda qanday elementlardan foydalanladi?
4. Vorislik mexanizmidan foydalanish uchun misol keltiring.
5. Quyidagi masalalar uchun bazaviy va hosila klasslarini tashkil qiling. Bu klasslardan amalda foydalanish uchun dastur ishlab chiqing.

№ var	Bazaviy klass va uning maydonlari	Hosila klass va uning maydonlari (bazaviy klass maydoni kursivda yozilgan)	Hosila klassi uchun metod
a)	sana (uchta son: yil, oy, kun)	do‘st: FISh, telefon, *tug‘ilgan kuni*	tug‘ilgan kungacha qolgan kunlar soni
b)	vaqt (uchta son): soat, minut, sekund)	uyali aloqa abonenti: familiyasi, operator, *joriy vaqt*	vaqtning imtiyozli ekanligini aniqlash (0 dan 8.00 gacha)
c)	to‘g‘ri kasr: surati, mahraji	aralash kasr: butun qismi, *surati* va *mahraji*	aralash kasrni haqiqiy son shaklida ifodalash

4-§. POLIMORFIZM

4.1. Umumiy ma’lumotlar

Polimorfizm so‘zini ko‘p yuzlama, ko‘p tomonlama kabi ma’nolarda tarjima qilish mumkin. Dasturlash tillarida esa ushbu atama klasslar tizimda bir xil nomga ega bo‘lgan metodlarga nisbatan qo‘llanadi.

Polimorflik xususiyati obyekt qaysi klassga mansub ekanligi haqidagi ma’lumotlar mavjud bo‘lmaganda ham uning klassini to‘g‘ri aniqlab, metodlarni aynan ana shu klass uchun bajarilishini ta’minlash bilan belgilanadi.

Obyektga asoslangan dasturlash nuqtayi nazaridan, bitta klass-dan (obyektdan) vorislik mexanizmi yordamida bir nechta hosila klasslarni yaratish mumkin. Tabiiyki, bu holda vorislik mexanizmiga ko‘ra, metodlarning nomlari bazaviy va hosila klasslarida bir xil bo‘ladi. Polimorfizm prinsipiga binoan u yoki bu metodga murojaat qilinganda u bilan bir xil nomga ega bo‘lgan boshqa metodlar emas, balki aynan nazarda tutilgan metod bajariladi.

Bazaviy klass obyektidan uning hosila klassidagi nomdosh obyektga o‘tish **toraytirish** deb ataladi va obyekt xususiyatlarini konkretlashtirish bilan birgalikda olib boriladi.

Hosila klassi obyektiga ko‘rsatkichni bazaviy klass obyektiga ko‘rsatkichga almashtirilsa, bazaviy klassda e’lon qilingan obyekt bilan ishlash imkoniyati paydo bo‘ladi. Bunday almashtirish **kengaytirish** deb ataladi.

Polimorfizm prinsipini qo‘llashdan maqsad klasslar uchun umumiy bo‘lgan xatti-harakatlarni amalga oshirishdan iborat. Har bir amalni bajarish ma’lumotlarning tipiga bog‘liq bo‘ladi. Masalan, absolyut qiymatni hisoblash uchun uchta turli funksiyalardan foyda-lanish mumkin: *abs(), labs()* va *fabs()*. Bu funksiyalar mos ravishda butun, katta butun va haqiqiy sonlarning absolyut qiymatini hisoblash uchun xizmat qiladi.

```
class A {
public: void    f1(){}
        };
class B : A {
public: void    f1(){}   // B klassda qayta aniqlash
        void    f2(){}
        };
```

```
void main()
{
B    x;
A  *pa = &x;     // to‘g‘ri almashtirish (kengaytirish) – oshkor
emas
pa->f1();        // B da turib  A::f1() ni chaqirish
pb = (B*) pa;    // teskari almashtirish (toraytirish) – oshkor
pb ->f2(); }     //agar pa B klass obyektini ko‘rsatsa to‘g‘ri
```

B klass obyektiga ko‘rsatkichni *A* klass obyektiga almashtirilganidan keyin ko‘rsatkich ostida *B* tursada, bazaviy klassdagi *A::f1()* funksiyasi chaqiriladi.

Ko‘rsatkich tipini almashtirish deganda xotira manzilini ko‘rsatkichda saqlash va uning talqinini o‘zgartirish tushuniladi. Bazaviy klassdan hosila klassga o‘tilganda bu xususiyat buzilishi mumkin.

C++ tilida funksiyani chaqirish vaqtidagi ma’lumot tipi aynan qaysi funksiyaning bajarilishini ko‘rsatib beradi. Bundan tashqari bitta funksiya nomidan turli harakatlarni amalga oshirish maqsadida ham foydalanish mumkin. Bu holni funksiyalarni qayta yuklash *(function overloading)* deb ataladi.

Polimorfizm prinsipini umumiy qilib, “bitta interfeys-ko‘pgina metodlar” deb aytish mumkin. Bu ma’nosi yaqin bo‘lgan amallar (harakatlar) uchun umumiy interfeys ishlab chiqish mumkinligini anglatadi.

Polimorfizm bitta sinfga taaluuqli bo‘lgan masalalar uchun yagona interfeysdan foydalanishga imkon beradi. Masalan, absolyut qiymatlarni hisoblashda bitta o‘ringa yuqorida keltirilgan uchta funksiyani qo‘llash dasturchi ishini murakkablashtirib yuboradi. Demak, ana shu funksiyalar uchun umumiy interfeys yaratish va polimorfizm prinsipiga ko‘ra foydalanish dasturning soddalashuviga sabab bo‘lishi mumkin.

Shuningdek, polimorfizm prinsipini operatorlarga nisbatan ham qo‘llash mumkin. Masalan, ma’lum bir ma’noda arifmetik amallarga nisbatan ham bu prinsipdan foydalaniladi. C++ tilida butun, katta

butun va harfiy ma’lumotlar uchun qo‘shish amalini misol sifatida keltirish mumkin. Bu amalni C++ tili kompilyatori avtomatik tarzda qaysi turdagi ma’lumotlarga nisbatan qo‘llanayotganligini aniqlaydi va shunga ko‘ra harakatni amalga oshiradi.

C++ tili dasturchilarga o‘zlari aniqlagan ma’lumot tiplari uchun ham qo‘llash imkonini beradi.

Demak, polimorfizm turli murakkablikdagi obyektlar uchun bir-biriga o‘xshab ketadigan amallar uchun umumiy interfeys ishlab chiqishni nazarda tutadi.

4.2. Virtual funksiyalar

Virtual funksiyalar *(virtual functions)* dinamik polimorfizmni (run − time polymorphism) qo‘llab-quvvatlagani uchun ham C++ tilida muhim hisoblanadi. Ma’lumki, polimorfizm prinsipidan ikki usulda foydalanish mumkin:

1) dasturni kompilatsiya qilish jarayonida funksiya va operatorlarni qayta yuklash vositasi sifatida;

2) dastur bajarilish vaqtida virtual funksiyalar vositasida qo‘llab-quvvatlanadi.

Virtual funksiya va dinamik polimorfizm asosini hosila klass obyektlariga ko‘rsatkichlar tashkil qiladi. C++ tilining o‘ziga xos tomonlaridan biri bazaviy klassga nisbatan aniqlangan ko‘rsatkichlardan ana shu klass hosilalariga nisbatan ham qo‘llash mumkinligi bilan belgilanadi. Shunga ko‘ra, quyidagi ko‘rsatmalar to‘g‘ri sanaladi:

```
base *p;              // bazaviy klassga ko‘rsatkich
base base_ob;         // bazaviy klass obyekti
derived derived_ob;   // hosila klass obyekti
r = &base_ob;         // bazaviy klass obyektiga ko‘rsatkich – p
r = &derived_ob;      // hosila klass obyektiga ko‘rsatkich – p
```

Yuqorida aytilganidek, bazaviy klass ko‘rsatkichidan shu klassdan hosil qilingan barcha hosilalardagi obyektlarni ko‘rsatish uchun foydalanish mumkin va bunda tiplar o‘rtasidagi nomutanosiblik

holati yuz bermaydi. Hosila klass obyektini koʻrsatish uchun bazaviy klass koʻrsatkichlaridan foydalanilganda ishlash uchun ruxsat faqat bazaviy klassdan vorislik orqali oʻtgan obyektlargagina beriladi. Buni shunday izohlash mumkin: bazaviy koʻrsatkich faqat oʻziga xos boʻlgan obyektlarini taniydi xolos, hosila klassiga qoʻshilgan yangi obyektlar haqida esa "hech narsani bilmaydi".

Bazaviy klass koʻrsatkichlarini hosila klass obyektlarini koʻrsatishda qoʻllash mumkin, ammo hosila klass koʻrsatkichidan bazaviy klass obyektlarini tavsiflash uchun foydalanish mumkin emas.

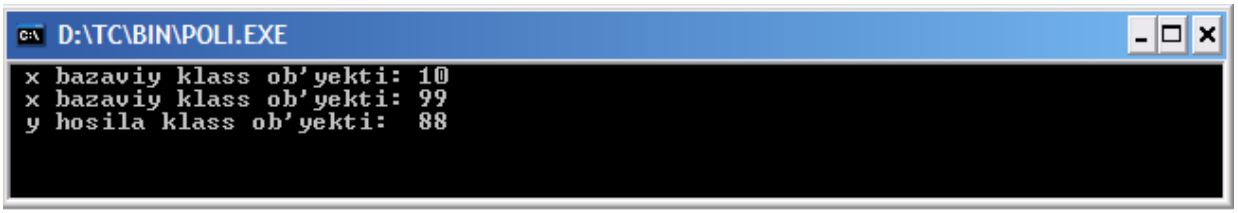
Shuni yodda tutish lozimki, koʻrsatkichlar arifmetikasi koʻrsatkichlarni eʼlon qilish vaqtida belgilab qoʻyigan maʼlumotlar tipi (klass) bilan chambarchas bogʻliq. Shunday qilib, agar bazaviy klass koʻrsatkichi hosila obyektini koʻrsatayotgan boʻlib, inkrementlashsa (qiymati birga ortsa), u holda bu koʻrsatkich hosila klassning navbatdagi obyektini emas, balki bazaviy klassning navbatdagi obyektini koʻrsatadi.

Quyidagi bazaviy klass koʻrsatkichini qay tarzda hosila klass obyektlariga nisbatan qoʻllash mumkinligi namoyish qilinadi.

```
#include <iostream.h>
class base
{
int x;
public:
void setx(int i) { x = I; }
int getx() { return x; }
};
class derived: public base { int y;
public :
void sety(int i) { y=I; }
int gety() {return y;}
} ;
int main()
{
```

```
base *p;          // bazaviy klass ko‘rsatkichi
base b_ob;        // bazaviy klass obyekti
derived d_ob;     // hosila klass obyekti
// p ko‘rsatkichdan bazaviy klass obyekti uchun foydalanish
p = &b_ob;
p->setx(10);       // bazaviy klass obyekti bilan ishlash
cout << “ x bazaviy klass obyekti: “ <<p->getx()<< ’\n’;
// p ko‘rsatkichdan hosila klass obyekti uchun foydalanish
p = &d_ob;        // hosila klass obyekti ko‘rsatilmoqda
p->setx(99);      //hosila klass obyekti bilan ishlash
//y uchun bu amalni to‘g‘ridan-to‘g‘ri bajaramiz
d_ob.sety (88) ;
cout << “x bazaviy klass obyekti: “ << p->getx() <<”\n”;
cout << “y hosila klass obyekti:  “ << d_ob.gety() << “\n”;
return 0;
}
```

Ushbu dastur quyidagi natijani beradi:



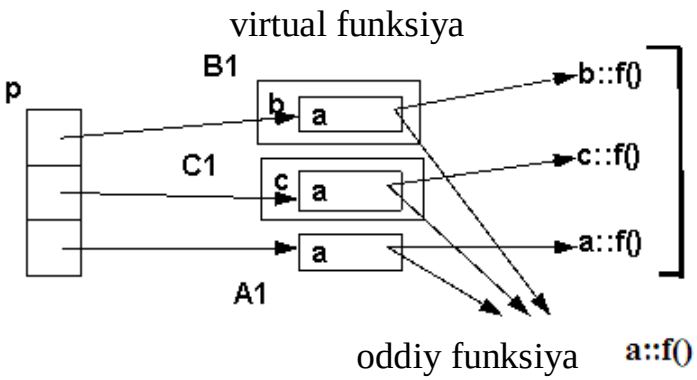
Qisqa qilib aytganda, virtual funksiya chaqirilganda obyekt tegishli bo‘lgan hosila klassgacha avtomatik torayish sodir bo‘ladi.

Virtual funksiyani keng ma’noda quyidagicha talqin qilish mumkin:

- virtual funksiya bazaviy klass uchun tavsiflanadi va hosila klassda vorislik hisobiga qayta aniqlanadi;
- funksiyani chaqirish bazaviy klass obyektga ko‘rsatkich orqali amalga oshiriladi va ixtiyoriy hosila klasslarning obyektlariga murojaat qilishi mumkin;
- virtual funksiyani chaqirish bazaviy klass obyektidan uning hosilasidagi obyektlarga o‘tadi va funksiyalarga murojaat aynan ularda sodir bo‘ladi.

Faraz qilaylik, *A* va uning *B*, *C* hosila klasslari e'lon qilingan bo'lsin. *A* klassda *f()* funksiya aniqlangan, *B* va *C* klasslarda esa bu funksiya vorislik prinsipiga ko'ra o'tgan va qayta aniqlangan bo'lsin. Bazaviy klass obyektlariga ko'rsatkichlar massivi *p* hamda u *A*, *B* va *C* klass obyektlariga ko'rsatkich sifatida initsializatsiya qilingan bo'lsin:

```
class a                 { virtual  void f(){} };
class b : public a      { ...   void f(){}   };
class c : public a      { ...   void f(){}   };
a  A1;       b  B1;     c C1;
a  *p[3] = { &B1, &C1, &A1 };
```



22-rasm. Virtual funksiya.

A bazaviy klass obyektiga ko'rsatkichning mavjudligi dasturning shu nuqtasida compilyator qaysi klass obyekti ko'rsatkich ostida turganligi haqidagi axborotga ega emasligini anglatadi. Shunday bo'lsada, agar funksiya virtual bo'lsa, unga bazaviy klass obyektiga ko'rsatkichga binoan murojaat qilinganda uning hosila klasslarini identifikatsiya qilishi va aynan ana shu klass uchun qayta aniqlangan funksiyani chaqirishi lozim:

```
p[0]->f();              //  B1 uchun  b::f() ga murojaat
p[1]->f();              //  C1 uchun  c::f() ga murojaat
p[2]->f();              //  A1 uchun  a::f() ga murojaat
for (i=0; i<2; i++)     //  obyekt tipiga mos ravishda
p[i]->f();              //  b::f(),c::f(),a::f() larga murojaat
```

Shunday qilib, “hosila klass ko‘rsatkichi” tipini “bazaviy klass ko‘rsatkichi” tipiga almashtirilganda hosila klass obyekti tipi haqidagi axborot yo‘qoladi.

4.3. Polimorfizmni amalda qo‘llashga misol

Faraz qilaylik, telefon kompaniyasi dasturchisi telefon bilan bog‘liq amallarni bajarish uchun dastur ishlab chiqishi talab qilingan bo‘lsin. Dasturda telefonga aloqador bo‘lgan nomer terish, qo‘ng‘iroq, uzish, bandlikni qayd etish kabi umumiy amallar nazarda tutilgan bo‘lsin. Bu amallarni o‘z ichiga oluvchi *phone* klassi ishlab chiqiladi:

```
class phone
{
public:
  void dial(char *number)
    { cout << “Nomer terish” <<  number << endl; }
  void answer(void) { cout << “javobni kutish”  << endl; }
  void hangup(void)
    { cout << “aloqa tugadi, trubkani osib qo‘ying “  << endl; }
  void ring(void) { cout << “Jiring, jiring, jiring “  << endl;}
  phone(char *number) { strcpy(phone::number, number); };
private:
  char number[13];
};
```

Quyidagi dastur telefon-obyektni hosil qilish uchun *phone* klassidan foydalanadi:

```
#include <iostream.h>
#include <string.h>
class phone
{
public:
  void dial(char *number)
    { cout << “Nomer terish” <<  number << endl; }
```

```
    void answer(void) { cout << "javobni kutish" << endl; }
    void hangup(void)
      { cout << "aloqa tugadi, trubkani osib qo'ying " << endl;
}
    void ring(void) { cout << "Jiring, jiring, jiring " << endl;}
    phone(char *number) { strcpy(phone::number, number); };
  private:
    char number[13];
  };
  void main(void)
  {
    phone telephone("555-1212");
    telephone.dial("212-555-1212");
  }
```

Bu dasturni ko'rgan boshliq dastur diskli va tugmachali telefonlarni farqiga bormasligi, pulli qo'ng'iroqlar uchun to'lovlarni hisobga olmaganligini bildirdi.

Dasturchi bunga javoban vorislik mexanizmi yordamida *phone* klassida *touch_tone* va *pay_phone* klasslarini hosil qiladi:

```
  class touch_tone : phone
  {
  public:
    void dial(char * number) { cout << "Pik pik Nomer terish"
    << number << endl; }
    touch_tone(char *number) : phone(number) { }
  };
  class pay_phone : phone
  {
  public:
    void dial(char * number)
    { cout << "Marhamat qilib " << amount << "sent to'lang"
     << endl;
```

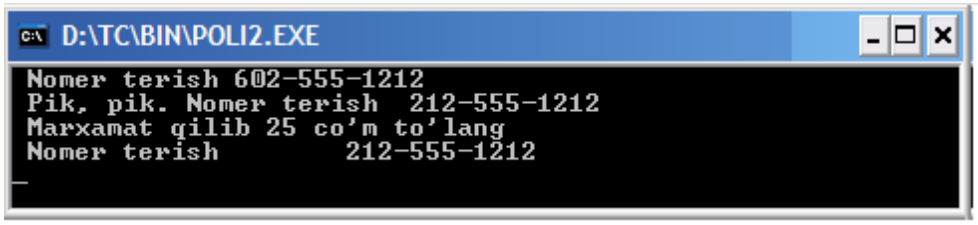
```
    cout << "Nomer terish "  << number  << endl;
  }
  pay_phone(char *number, int amount) : phone(number)
    {  pay_phone::amount = amount; }
 private:
  int amount;
 };
```

Ko‘rib turibsizki, *touch_tone* va *pay_phone* klasslari uchun shaxsiy *dial* metodi tashkil qilingan. Quyidagi dasturda bu klasslardan *rotary, touch_tone* hamda *pay_phone* obyektlarini yaratish uchun foydalanilgan:

```
#include <iostream.h>
#include <string.h>
class phone
{
public:
  void dial(char *number) { cout << " Nomer terish " <<
  number << endl; }
   void answer(void) { cout << " Javobni kutish" << endl; }
   void hangup(void) { cout << " Qo‘ng‘iroq tugadi, trubkani ilib
qo‘ying" << endl; }
  void ring(void) { cout << "Jiring, jiring, jiring" << endl; }
  phone(char *number) { strcpy(phone::number, number); };
protected:
  char number[13];
};
class touch_tone : phone
{
public:
   void dial(char *number) { cout << " Pik, pik. Nomer terish  "
   << number << endl; }
   touch_tone(char *number) : phone(number) { }
};
```

```
class pay_phone : phone
{
public:
  void dial(char * number) { cout << " Marhamat qilib " <<
  amount << " co‘m to‘lang " << endl;
  cout << " Nomer terish " << number << endl; }
   pay_phone(char * number, int amount) : phone(number)
  { pay_phone::amount = amount; }
private:
   int amount ;
};
void main (void)
{
   phone rotary(“303-555-1212”);
   rotary.dial(“602-555-1212”);
   touch_tone telephone(“555-1212”);
   telephone.dial(“212-555-1212”);
   pay_phone city_phone(“555-1111”, 25);
  city_phone.dial(“212-555-1212”);
 }
```

Ushbu dastur quyidagi natijani ekranga chiqaradi:



Yuqorida ta’kidladikki, polimorf obyekt dasturning bajarilishi vaqtida o‘z shaklini o‘zgartirishi mumkin.

4.4. Polimorf obyekt-telefon ishlab chiqish

Endi yuqoridagi dasturga qo‘shimcha tugmali yoki diskli telefonni ajratib olish masalasi ham qo‘yilgan bo‘lsin. Boshqacha aytganda, dastur mustaqil ravishda qo‘ng‘iroq qanday telefondan

qilinganligiga qarab, yoki pullik telefon, yoki tugmali telefon ekanligini ajratishi va shunga mos javob reaksiyasini koʻrsatishi lozim. Sodda qilib aytganda, qoʻngʻiroqqa qarab oʻz shaklini oʻzgartirishi talab qilinadi.

Telefonning bunday turli klasslari bir-biridan faqt bitta *dial* metodi bilan farq qiladi. Polimorf obyekt yaratish uchun dastlab bazaviy klass funksiyalari ishlab chiqiladi. Bu funksiyalar hosila klass funksiyalaridan virtualligi bilan farq qiladi va ularning prototiplari *virtual* xizmatchi soʻzi yordamida belgilab qoʻyiladi:

```
class phone
{
public:
   virtual void dial(char *number)
   { cout << " Nomer terish " << number << endl; }
   void answer(void) { cout << " Jabobni kutish " << endl; }
   void hangup(void)
  {cout << " Qo'ngiroq tugadi, trubkani ilib qo'ying " << endl;
}
   void ring(void) { cout << "Jiring, jiring, jiring " << endl; }
   phone(char *number) { strcpy(phone::number, number); };
protected:
   char number[13];
};
```

Shundan keyin bazaviy klass obyektiga koʻrsatkich yaratiladi. Buning uchun *phone* bazaviy klassiga koʻrsatkich kiritiladi:

```
phone *poly_phone;
```

Obyekt shaklini oʻzgartirish uchun bu koʻrsatkichga hosila klass obyekti adresini quyidagi koʻrinishda oʻzlashtiriladi:

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

Oʻzlashtirish operatoridan keyin yozilgan (*phone* *) yozuvi tiplarni oʻzgartirish operatori boʻlib hizmat qiladi va kompilyatorga

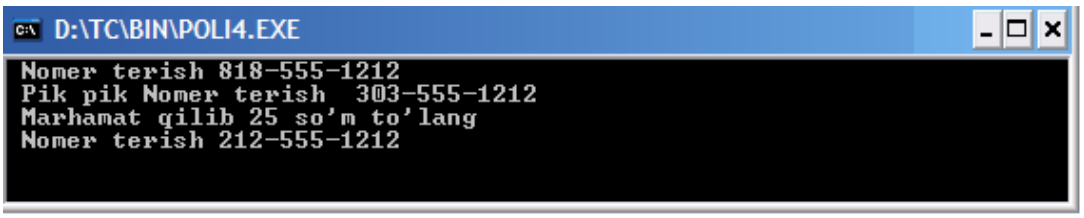
“hamma narsa joyida” degan ma’noda xabar berib, bir tipdagi (*touch_tone*) o‘zgaruvchi adresini boshqa tipdagi (*phone*) o‘zgaruvchi ko‘rsatkichiga o‘zlashtiradi. Dastur *poly_phone* obyekti ko‘rsatkichiga turli obyektlar adresini o‘zlashtirgani uchun bu obyekt o‘z shaklini o‘zgartiradi va demak, u polimorf bo‘ladi.

Quyidagi dastur ana shunday polimorf obyekt-telefon yaratish uchun mo‘ljallangan. Dastur ishga tushganidan keyin diskli telefon tugmaliga, so‘ngra pullik telefonga aylanadi:

```
#include <iostream.h>
#include <string.h>
class phone
{
public:
   virtual void dial(char *number)
   { cout << “ Nomer terish “ << number << endl; }
   void answer(void) { cout << “ Jabobni kutish ” << endl; }
   void hangup(void)
  { cout << “ Qo‘ngiroq tugadi, trubkani ilib qo‘ying ” << endl;
}
   void ring(void) { cout << “Jiring, jiring, jiring “ << endl; }
   phone(char *number) { strcpy(phone::number, number); };
protected:
   char number[13] ;
};
class touch_tone : phone
{
public:
   void dial(char * number) { cout << “ Pik pik Nomer terish “
<< number << endl; }
   touch_tone(char *number) : phone(number) { }
};
class pay_phone: phone
```

```
{
public:
   void dial(char *number) { cout << “Marhamat qilib “ <<
  amount << “ so‘m to‘lang”  << endl;
  cout << “ Nomer terish “       << 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”) ;
     // Obyektni diskli telefonga aylantirish
   phone *poly_phone = &rotary;
   poly_phone->dial(“818-555-1212”);
     // Obyekt shaklini tugmali telefonga aylantirish
   poly_phone = (phone *) &home_phone;
   poly_phone->dial(“303-555-1212”);
     // Obyekt shaklini pullik telefonga aylantirish
   poly_phone = (phone *) &city_phone;
   poly_phone->dial(“212-555-1212”);
}
```

Bu dastur ekranga quyidagi ma’lumotlarni chiqadi:



Dastur *poly_phone* obyekti o‘z shaklini o‘zgartirayotgani uchun polimorf hisoblanadi.

Polimorf obyektlarni yaratish uchun dasturda bazaviy klass obyektiga koʻrsatkichdan foydalaniladi. Soʻngra dastur bu koʻrsatkichga hosila klass obyekti adresini oʻzlashtiradi. Har gal dastur koʻrsatkichga boshqa hosila klass obyekti adresini koʻrsatganda, bu koʻrsatkichning obyekti oʻz shaklini oʻzgartiradi, ya'ni polimorflik sodir boʻladi.

Polimorf obyekt yaratish uchun dastur bazaviy klassning bir yoki bir nechta metodlarini virtual funksiya tarzida aniqlashi lozim. Hosila klasslar esa bazaviy klassning virtual funksiyasi sifatida ishlaydigan oʻz shaxsiy funksiyalarini qurishlari kerak.

Sof virtual funksiya qurish uchun funksiyaning prototipi koʻrsatiladi, ammo operatorlari keltirilmaydi. Ularning oʻrniga funksiya quyida koʻrsatilganidek nol qiymatini qoʻyadi:

```
class phone
{
public:
  virtual void dial (char *number) =0; // sof virtual funksiya
  void answer(void) { cout << " Javobni kutish "  << endl; }
  void hangup(void) { cout << " Qo'ng'iroq tugadi, trubkani
  ilib  qo'ying " << endl; }
  void ring(void) { cout << " Jiring, jiring, jiring " << endl; }
  phone(char *number) { strcpy(phone::number, number); };
protected:
  char number[13];
};
```

Takrorlash uchun savol va topshiriqlar

1. Polimorfizm prinsipi deganda nimani tushunasiz?
2. Virtual va novirtual funksiyalar nima?
3. Obyektning polimorflik hodisasi qachon roʻy beradi?
4. Quyidagi masalalar uchun dastur ishlab chiqing. Bunda polimorfizm prinsipiga asoslaning.

a) Bazaviy klass: kompleks son: sonning haqiqiy va mavhum qismlari (*a1*, *b1*). Hosila klass maydonlari: ikkita kompleks son

maydonlari (mos ravishda (*a1*, *b1*) hamda (*a2*, *b2*). Hosila klassida ikki kompleks sonning ko‘paytmasini hisoblang.

b) Bazaviy klass: kitob: nomi, betlari soni, narxi. Hosila klass: kutubxona: nomi, betlari soni, arzonlashtirish foizi. Hosila klassida kitobning arzonlashganidan keyingi narxini hisoblang.

c) Bazaviy klass: vaqt: soat, minut va sekund. Hosila klass: dars jadvali: fan, boshlanish vaqti, auditoriya. Hosila klassida ko‘rsatilgan vaqtda qaysi fan ekanligi aniqlansin.

5-§. ISTISNO QILINADIGAN HOLATLARNI QAYTA ISHLASH

5.1. Xatoliklarni qayta ishlashning umumiy mexanizmi

Ayrim hollarda dastur kutilmagan bir qator vaziyatlar ro‘y berganda o‘z vazifasini bajara olmay qolishi mumkin. Nolga bo‘linish, mavjud bo‘lmagan xotira qismiga murojaat qilish mumkin bo‘lgan ana shunday holatlardan sanaladi.

Istisno qilinadigan holatlar (sodda qilib xatoliklar deb aytish mumkin) – bajarilishi jarayonida oldindan kutilmagan holatlar yuzaga kelganda dastur o‘z ishini davom ettirishini ta’minlashdan iborat.

C++ tili istisno qilinadigan holatlar sodir bo‘lganda dastur o‘zini qanday tutishini belgilab qo‘yish uchun dasturchilarga bir qator vositalarni taklif qiladi.

Ma’lumki, xatoliklar ikki turga bo‘linadi: kompilatsiya vaqtidagi va dasturni bajarish vaqtidagi xatoliklar. Odatda 1-turdagi xatoliklarni kompilyator aniqlab beradi, 2-tur xatoliklarni esa faqat dastur bajarilayotgan vaqtda aniqlash mumkin, xolos. 2-tur xatoliklar yuzaga kelganda dastur o‘z ishini to‘xtatib qo‘yadi. Dasturchi buning oldini olishi, ya’ni dastur har qanday holda ham o‘z ishini davom ettirib, kutilgan natijani berishini ta’minlashi lozim. Boshqacha aytganda, bajarish vaqtida yuzaga kelishi mumkin bo‘lgan turli xatoliklarni nazarda tutishi va ularning har biri sodir bo‘lganda dasturning javob reaksiyasini belgilab qo‘yishi talab qilinadi.

Dastur ishlab chiqish jarayonida dasturchi xatoliklar yuzaga kelishi mumkin bo‘lgan barcha holatlarni nazorat qilishi lozim. Bunday holatlar *try* bilan boshlanadigan nazorat bloklarida qayta ishlanadi.

Istisno qilinadigan holatlarni qayta ishlash xato yuzaga kelganidan keyin boshlanadi. Bu jarayonni tashkil qilish uchun *throw* operatoridan foydalaniladi.

Dasturning javob reaksiyasi xatolikni qayta ishlagichlar yordamida ta’minlanadi. Agar dasturning ularga mos javob reaksiyasi belgilanmagan bo‘lsa, standart *terminate* funksiyasi ishga tushadi va u hisoblash jarayonini to‘xtatish uchun *abort* funksiyasini chaqiradi. Dasturchi mana shunday hollarda jarayonni to‘xtatish uchun shaxsiy funksiyalarini ishlab chiqishi mumkin.

5.2. Istisno qilinadigan holatlar sintaksisi

Istisno qilinadigan holatlarni nazorat qiluvchi blok *try* so‘zi bilan boshlanadi va figurali qavslar orasida yoziladi.

```
Try
{
...
}
```

Istisno qilinadigan holatlarni aniqlash *throw* xizmatchi so‘zi yordamida amalga oshiriladi:

```
throw [ ifoda ];
```

Throw dan keyin ko‘rsatilgan ifodaning tipi xatolik tipini aniqlab beradi. Xatolikni qayd etish vaqtida joriy blokni bajarish jarayoni to‘xhtatiladi va boshqaruv unga mos qayta ishlagichga uzatiladi.

Yuzaga kelgan xatolikni to‘g‘ri qayta ishlash har doim ham mumkin bo‘lavermaydi. Ayrim hollarda bir nazorat blogi ikkinchisining ichida kelishi mumkin. Bunday holda boshqaruv parametrsiz *throw* yordamida tashqi qayta ishlagichlarga uzatiladi.

Xatoliklarni qayta ishlagichlar *catch* xizmatchi so‘zi bilan boshlanib, qavslar ichida istisno qilinadigan holatlar tipi ko‘rsatiladi. Ular

bevosita *try* blogidan keyin yoziladi. Qayta ishlanayotgan xatolik-larning tipiga mos ravishda bir yoki bir nechta qayta ishlagichlardan foydalanish mumkin.

Qayta ishlagichlarni umumiy holda quyidagi koʻrinishlardan birida yozish mumkin:

```
catch(tip  nom){ ... /* qayta ishlagich*/ }
catch(tip){ ... /* qayta ishlagich */ }
catch(...){ ... /* qayta ishlagich  */ }
```

1-shakl parametr nomi qayta ishlagichda qandaydir amallarni bajarishni tashkil qilishga toʻgʻri kelganda (masalan, xatolik haqida axborotni ekranga chiqarilganda) qoʻllanadi. Ikkinchi shakl esa xatolik haqida axborot berishni nazarda tutmaydi. Uchinchi shakldagi uch nuqta qayta ishlagich hamma xatoliklarni tutib qolishini anglatadi. Masalan:

```
catch (int i)
{
... // int tipidagi xatolikni qayta ishlash
}
catch (const char *)
{
... // const char* tipidagi xatolikni qayta ishlash
}
catch (overflow)
{
... // Overflow klassidagi xatoliklarni qayta ishlash
}
catch(...)
{
... // koʻzda tutilmagan barcha xatoliklarni qayta ishlash
}
```

Qayta ishlash tugaganidan so‘ng boshqaruv bevosita qayta ishlagichdan keyin ko‘rsatilgan birinchi operatorga uzatiladi. *Try* blogida ko‘rsatilgan xatoliklar ro‘y bermaganda ham boshqaruv aynan shu operatorga o‘tadi.

5.3. Xatoliklarni tutib qolish

Throw operatori yordamida istisno qilinadigan holatlar qayd qilinganda C++ quyidagi amallarni bajaradi:

1) *throw* parametrini statik obyekt sifatida nusxasini oladi va istisno qilinadigan holatlar qayta ishlanmaguncha saqlab turadi;

2) mos qayta ishlagichni qidirib, steklarni aylantirib ko‘rib chiqadi va amal qilish doirasidan chetga chiqqan lokal obyektlarning destruktorlarini chaqiradi;

3) boshqaruvni shu obyekt bilan tipi bir xil bo‘lgan parametrli qayta ishlagichga uzatadi.

Qayta ishlagich topilgan hisoblanadi, agar *throw* dan keyin ko‘rsatilgan obyektning tipi:

a) catch ning parametrida ko‘rsatilgan bo‘lsa (parametr T, const T, T& yoki const T& shaklida yozilgan bo‘lshi mumkin. Bu yerda T − xatolik tipi);

b) catch parametrining hosilasi bo‘lsa (agar vorislik *public* kaliti bilan hosil qilingan bo‘lsa);

c) tipini standart tiplarni almashtirish qoidalari yordamida *catch* parametridagi ko‘rsatkich tipiga keltirish mumkin bo‘lgan ko‘rsatkichlar.

Ko‘rinib turibdiki, klass hosilalari qayta ishlagichlarini bazaviy qayta ishlagichlardan oldinroq joylashtirish lozim, aks holda boshqaruv hech qachon ularga o‘tmaydi. *Void* tipidagi ko‘rsatkichlarni qayta ishlagichlar to‘g‘ridan-to‘g‘ri boshqa tipdagi ko‘rsatkichlarni to‘sib qo‘yadi va shu sababli uni barcha konkret tipli qayta ishlagichlardan keyin ko‘rsatish lozim.

Quyidagi dasturga e’tibor bering.

```
#include <iostream.h>
class Hello
{
// o‘zining yo‘qotilgani haqida axborot beruvchi klass
public:
Hello(){cout <<”Hello!” << endl;}
~Hello(){cout << “Bye!” << endl;}
}:
void fl()
{
ifstream ifs(“\\INVALID\\FILE\\NAME”);  // Faylni ochamiz
if (!ifs) {
cout <<“xatolikni qayd etamiz << endl;
throw “Fayli ochishdagi xatolik ”;}
}
void f2()
{
Hello H; // Lokal obyekt yaratilmoqda
fl();   // xatolikni yuzaga keltiruvchi finksiya chaqirilmoqda
}
int main()
{
Try
{
cout << “ try-blokka kirish” << endl;
f2();
cout << “try-blokdan chiqish “ << endl;
}
catchdnt i)
{
cout << “int. istisnoni qayta ishlagich chaqirildi- “ << I << endl;
return -1;
}
```

```
catch(const char * p)
{
cout <<“const char* istisnoni qayta ishlagich chaqirildi-“<< p
<< endl;
return -1;
}
catch(...)
{
cout << “Barcha istisnolarni qayta ishlagich chaqirildi – “<< endl;
return -1;
}
return 0; // Hammasi yaxshilik bilan tugadi
}
```

Ushbu dastur quyidagi natijani beradi:

```
Try – blokka kirish
Hello!
Istisnoni qayd qilamiz
Bye!
Const char* istisnoni qayta ishlagich chaqirildi – Faylni
ochishdagi xatolik
```

E’tibor bering, xatolik yuz berganidan keyin lokal obyektning destruktori chaqirildi, ammo bu vaqtda boshqaruv *f1* dan *main* funksiyasida turgan qayta ishlagichga uzatildi. “Try-blokdan chiqish” axboroti ekranga chiqarilmadi. Dasturda fayllar bilan ishlash uchun oqimlardan foydalanildi.

Shunday qilib, istisnolarni qayta ishlash mexanizmi xatoliklar yuz berganda obyektlarni yo‘qotishi mumkin. Shuning uchun resurslarni ajratish va bo‘shatish amalini klasslar ko‘rinishida (konstruktor tashkil qiladi, destruktor esa bo‘shatadi) tashkil qilish maqsadga muvofiq hisoblanadi. Misol tariqasida fayllar bilan ishlash uchun klassni keltirish mumkin. Bu klassning konstruktori faylni ochadi, destruktor esa yopadi. Albatta bu holda xatolik yuz berganda faylni yopish to‘g‘ri tashkil qilinadi va undagi ma’lumotlar yo‘qolmaydi.

Ta’kidlab o‘tilganidek, istisno qilinadigan holatlar standart tipda ham, foydalanuvchi aniqlagan tipda ham bo‘lishi mumkin. Bunday hollarda bu tipni global e’lon qilish shart emas va xatolikni qayd qilish hamda ularni qayta ishlash vaqtida ma’lum bo‘lsa yetarli.

Istisno qilinadigan holatlarni ifodalovchi klasslarni istisnolarni qayta ishlash vaqtida yuzaga kelishi mumkin bo‘lgan klasslar ichida e’lon qilish mumkin. Bu klassning ko‘chirish konstruktori *public* tarzida e’lon qilinishi shart, aks holda xatolikni qayd qilish vaqtida obyektning nusxasini yaratish mumkin bo‘lmay qoladi.

5.4. Konstruktor va destruktordagi istisnolar

C++ tili konstruktor va destruktordan qiymat qaytarishda foydalanishga ruxsat bermaydi. Istisnolarni qayta ishlash mexanizmi obyektning konstruktori yoki destruktorida yuzaga kelgan xatolik haqida axborot berishi mumkin. Bu fikrni namoyish qilish uchun *Vector* klassini tashkil qilamiz. Unda so‘raladigan xotira hajmi cheklanadi.

```
Class Vector{
public−
class Size{};                    // istisno klassi
enum {max = 32000}:              // vektorning maksimal uzunligi
Vector(int n)                    // konstruktor
{ if (n<0 || n>max) throw SizeO: ... }
};
```

Vector klassidan foydalanganda *Size* tipidagi xatoliklarni kuzatish mumkin:

```
try{
Vector *p = new Vector(i):
}
catch(Vector::Size){
...   // vector o‘lchami bilan bog‘lis xatolikni qayta ishlash
}
```

Qayta ishlagichda xatolik haqida axborot berish va qayta tiklashning asosiy usullaridan foydalanish mumkin. Istisnoni aniqlovchi klass ichida qayta ishlagichga uzatiladigan istisno haqidagi axborotni ham saqlashga ruxsat beriladi. Buning ma’nosi istisno qilinadigan holat aniqlangan nuqtadan xatolik haqidagi axborotni qayta ishlagich yetarlicha imkoniyatga ega bo‘lgan joyga uzatishni ta’minlashdan iborat.

Agar obyekt konstruktorida istisno qayd qilinsa, avtomatik tarzda joriy vaqtgacha shu blokda to‘la yaratilgan obyektlar hamda joriy obyektning maydonlari uchun destruktor chaqiriladi. Masalan, agar istisno obyektlar massivini yaratishda yuzaga kelsa, destruktorlar faqat muvaffaqiyatli yaratilgan elementlar uchun ishga tushadi.

Agar obyekt dinamik xotirada *new* yordamida yaratilayotgan bo‘lib, konstruktorda xatolik ro‘y bersa, bu obyekt band qilgan xotira qismi bo‘shatiladi.

Standart tipdagi istisnolarga qaraganda shaxsiy tipdagi istisnolarni qayta ishlash afzal sanaladi. Istisnolarni qayta ishlashga qaraganda klasslar yordamida istisnolar haqidagi axborotlarni uzatish osonroq. Bundan tashqari, klasslar shajarasidan foydalanishga imkon paydo bo‘ladi.

Istisnolarni boshqarish mexanizmi bazaviy klasslar uchun shaxsiy qayta ishlagichlarni yaratishga imkon beradi, qardosh (bir-biriga yaqin) istisnolarni ko‘pincha shajaralar ko‘rinishida yaratish mumkin bo‘ladi. Umumiy bazaviy klassdan istisnolarni keltirib chiqarishda polimorfizm prinsipidan foydalanib qayta ishlagichda bazaviy klassga ko‘rsatkichlarni tutib qolish mumkin bo‘ladi. Masalan, mataematik kutubxonada klasslarni quyidagicha tashkil qilish mumkin:

```
class Matherr{};
class Overflow: public Matherr{};       // xotiraning to‘lib ketishi
class Underflow: public Matherr{};      // tartibning yo‘qolishi
class ZeroDivide: public Matherr{};     // nolga bo‘linish
```

Ma’lumotlarni kiritish va chiqarish bilan bog‘liq xatoliklarni ifodalash uchun quyidagi klasslardan foydalanish mumkin:

```
class  Ioegg{}:
class  Readerr:  public  Ioerg{};        //o‘qishdagi xatolik
class  Writerr:  public  Ioegg{}:        // yozishdagi xatolik
class  Seekerr:      public  Ioegg{};    // qidirishdagi xatolik
```

Vaziyatga bog‘liq ravishda yoki hosila istisnolarni ham tutib qola oladigan bazaviy klass qayta ishlagichidan yoki shaxsiy hosila klass qayta ishlagichlaridan foydalanish mumkin.

ILOVA

1-ilova. ASCII (ruslashtirilgan) jadvali

Kod	Belgi	Kod	Belgi	Kod	Belgi	Kod	Belgi	Kod	Belgi	Kod	Belgi	Kod	Belgi	Kod	Belgi
0		32		64	@	96	‘	128	А	160	а	192	└	224	р
1	☺	33	!	65	A	97	a	129	Б	161	б	193	┴	225	с
2	☻	34	“	66	B	98	b	130	В	162	в	194	┬	226	т
3	♥	35	#	67	C	99	c	131	Г	163	г	195	├	227	у
4	♦	36	$	68	D	100	d	132	Д	164	д	196	─	228	ф
5	♣	37	%	69	E	101	e	133	Е	165	е	197	┼	229	х
6	♠	38	&	70	F	102	f	134	Ж	166	ж	198	╞	230	ц
7		39	’	71	G	103	g	135	З	167	з	199	╟	231	ч
8		40	(	72	H	104	h	136	И	168	и	200	╚	232	ш
9	○	41	)	73	I	105	i	137	Й	169	й	201	╔	233	щ
10		42	*	74	I	106	i	138	К	170	к	202	╩	234	ъ
11	♂	43	+	75	K	107	k	139	Л	171	л	203	╦	235	ы
12	♀	44	,	76	L	108	l	140	М	172	м	204	╠	236	ь
13		45	-	77	M	109	m	141	Н	173	н	205	═	237	э
14	♫	46	.	78	N	110	n	142	О	174	о	206	╬	238	ю
15	☼	47	/	79	O	111	o	143	П	175	п	207	╧	239	я
16	►	48	0	80	P	112	p	144	Р	176	░	208	╨	240	Ё
17	◄	49	1	81	Q	113	q	145	С	177	▒	209	╤	241	ё
18	↕	50	2	82	R	114	r	146	Т	178	▓	210	╥	242	Є
19	‼	51	3	83	S	115	s	147	У	179	│	211	╙	243	є
20	¶	52	4	84	T	116	t	148	Ф	180	┤	212	╘	244	Ї
21	§	53	5	85	U	117	u	149	Х	181	╡	213	╒	245	ї
22	▬	54	6	86	I	118	v	150	Ц	182	╢	214	╓	246	Ў
23	↨	55	7	87	W	119	w	151	Ч	183	╖	215	╫	247	ў
24	↑	56	8	88	X	120	x	152	Ш	184	╕	216	╪	248	°
25	↓	57	9	89	Y	121	y	153	Щ	185	╣	217	┘	249	·
26	→	58	:	90	Z	122	z	154	Ъ	186	║	218	┌	250	·
27	←	59	;	91	[	123	{	155	Ы	187	╗	219	█	251	√
28	∟	60	<	92	\	124	|	156	Ь	188	╝	220	▄	252	№
29	↔	61	=	93	]	125	}	157	Э	189		221	▌	253	¤
30	▲	62	>	94	^	126	~	158	Ю	190	╜	222	▐	254	□
31	▼	63	?	95	_	127	⌂	159	Я	191	┐	223	▀	255	

2-ilova. Math.h modulidagi ayrim funksiyalar

Funksiya prototipi	Bajaradigan amali
int abs(int i)	*I* sonning absolut qiymatini qaytaradi
double acos(double x)	radianda berilgan *x* argument arkkosinusi
double asin(double x)	radianda berilgan *x* argument arksinusi
double atan(double x)	radianda berilgan *x* argument arktangensi
double atan2(double x, double y)	radianda berilgan *x/y* nisbatning arktangensi
double ceil(double x)	haqiqiy *x* ga eng yaqin bo‘lgan katta butun sonning haqiqiy son ko‘rinishdagi ifodasi
double cos(double x)	*x* radianli burchak kosinusi
double cosh(double x)	*x* radianli burchakning giperbolik kosinusi
double exp(double x)	e^x qiymatni qaytaradi
double fabs(double x)	*x* haqiqiy sonning absolut qiymati
double floor(double x)	haqiqiy *x* ga eng yaqin kichik butun sonning haqiqiy son ko‘rinishidagi ifodasi
double fmod(*double x, double y*)	*x* sonini *y* soniga bo‘lib, haqiqiy son ko‘rinishidagi qoldiqni qaytaradi
double hypot(double x, double y)	to‘g‘ri burchakli uchburchakning gipotenuzasi

long int labs(long int num)	*num* uzun butun sonning absolut qiymati
double ldexp(*double x, int exp*)	$x*2^{exp}$ qiymatni qaytaradi
double log(*double x*)	*x* soni natural logarifmi
*double log*10(*double x*)	*x* sonining 10 asosli logarifmi
double modf(*double x. double *intptr)*	*x* sonining kasr qismini qaytaradi va butun qismini *intptr* adresga joylaydi
double poly(*double x, int n, double c*[])	$c[n]x^n + c[n-1]x^{n-1} + ... + c[1]x + c[0]$ ko‘phadning qiymatini hisoblaydi
double pow(*double x, double y*)	x^y ni hisoblaydi
*double pow*10(*int p*)	10^p ni hisoblaydi
double sin(double x)	*x* radianli burchak sinusi
double sinh(double x)	*x* radianli burchakning giperbolik sinusi
double sqrt(double x)	*x* soninnng kvadrat ildizi
double tan(double x)	*x* radianli burchakning giperbolik kosinusi

FOYDALANILGAN ADABIYOTLAR RO‘YXATI

1. *Aripov M.M. Programmalashga kirish*. – T.: O‘zMU, 2008.
2. *Aripov M.M.* C++ tiliga kirish. – T.: O‘zMU, 2007. -172-b.
3. *Aripov M.M. Otaxanov N.A.* Dasturlash asoslari bo‘yicha masalalar to‘plami. – Namangan: “Ibrat”, 2014. -168-b.
4. *Бондарев В.М.* Программирование на C++. –Харьков: “Компания Смит”, 2005. -284 с.
5. *Madrahimov Sh.F., G‘aynazarov S.M.* C++ tilida programmalash asoslari. – T.: O‘zMU, 2009. -196-b.
6. *Павловская Т.А.* C/C++ программирования на высоком уровне. – СПБ: Питер, 2003. -462 с.
7. *Подбельский В.В.* Язык СИ++. – М.: “Финансы и статситика”, 2003. -560 с.
8. *Стивен Пратта.* Программирование на C++. – СПБ: “ООО ДиасофтЮП”, 2005. -1104 с.
9. *Строуструп Б.* Язык программирования C++. Специальное издание. – Нью Джерси: “Аддисон Веслей”, 2009. -1054 с.
10. *Харви М. Дейтел, Пол Дж. Дейтел*. Как программировать на C++. – М.: “Бином”, 2008. -1454 с.
11. *Шильд Герберт*. C++. Базовый курс. – М.: Издательский дом “Вильямс”, 2010. -624 с.
12. Уроки по C++. Урок №23. Конструктор и деструктор. http://www. programmersclub.ru/23/

MUNDARIJA

Aripov Mirsaid Mirsiddiqovich

Otaxanov Nurillo Abdumalikovich

DASTURLASH ASOSLARI

Oliy o‘quv yurtlarining matematika, amaliy matematika va informatika bakalavr yo‘nalishi talabalari uchun o‘quv qo‘llanma

«TAFAKKUR BO‘STONI»
TOSHKENT — 2015

Muharrir	*Sh. Rahimqoriyev*
Musahhih	*S. Abduvaliyev*
Tex. muharrir	*D. O‘rinova*
Sahifalovchi	*U. Vohidov*

Litsenziya AI № 190, 10.05.2011-y.

Bosishga 2015-yil 10-oktabrda ruxsat etildi. Bichimi 60S84$^{1}/_{16}$.
Ofset qog‘ozi. «Times» garniturasi. Shartli bosma tabog‘i 14,0.
Nashr tabog‘i 14,5. Shartnoma №. Adadi 500. Buyurtma №.

«TAFAKKUR BO‘STONI» MCHJ.
100190, Toshkent shahri, Yunusobod tumani, 9-mavze, 13-uy.
Telefon: 199-84-09. E-mail: tafakkur0880@mail.ru

«TAFAKKUR BO‘STONI» MCHJ bosmaxonasida chop etildi.
Toshkent shahri, Chilonzor ko‘chasi, 1-uy.