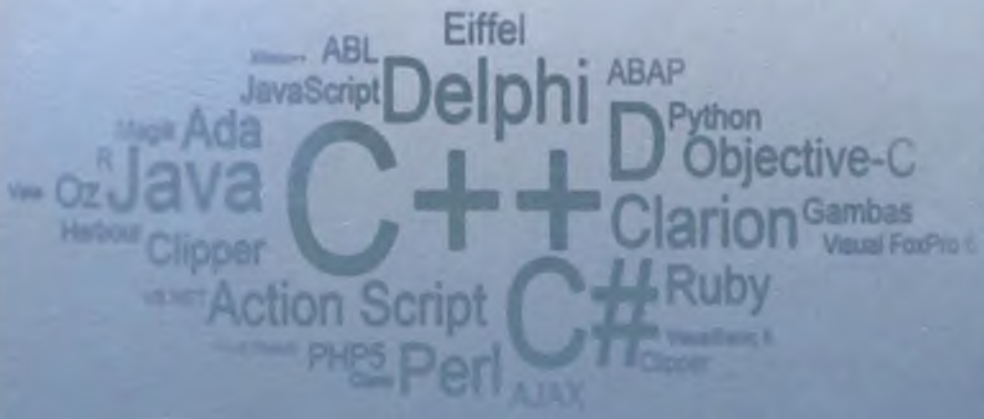
T.A.Maxarov

DASTURLASH ASOSLARI

Visual Studio



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

T. A. Maxarov

DASTURLASH ASOSLARI

(Visual Studio)
(o‘quv qo‘llanma)

Toshkent
«IJOD-PRESS»
2019

UO'K: 004.42(075.8)
KBK: 32.973.26-018.2ya73
M32

M32 **Maxarov T.A.**
Dasturlash asoslari (Visual Studio). / Bakalavrlar uchun o'quv qo'llanma: – T.: «IJOD-PRESS» nashriyoti, 2019-yil, – 248 bet.

Taqrizchilar: Mirzo Ulug'bek nomidagi O'zbekiston Milliy Universiteti Matematika fakulteti "Amaliy matemayika va kompyuter tahlili" kafedrasi dotsenti, fizika-matematika fanlari doktori **Z.R.Raxmonov**

Toshkent Axborot texnologiyalari universiteti qoshidagi Dasturiy mahsulotlar va apparat-dasturiy majmualar yaratish markazi yetakchi ilmiy xodimi, texnika fanlari doktori, professor **D.T.Muxammadiyeva**

O'quv qo'llanma 330000 – «Kompyuter texnologiyalari va informatika» ta'lim sohasi yo'nalishlarida, 5130100 – «Matematika», 5130200 – «Amaliy matermatika va informatika», 5140300 – «Mexanika» bakalavriat ta'lim yo'nalishlarida, hamda mustaqil o'rganuvchilar uchun mo'ljallangan. Qo'llanma «Dasturlash asoslari» fanining asosiy mavzularini qamrab olgan bo'lib, talabalar fanni o'zlashtirishda foydalanishlari mumkin.

UO'K: 004.42(075.8)
KBK: 32.973.26-018.2ya73

ISBN 978-9943-5815-8-6

MUNDARIJA

KIRISH

Butun dunyoda axborot texnologiyalarining rivojlanishi mamlakat farovonligi va iqtisodiy o'sishning asosiy omillaridan biriga aylanib bormoqda. Shu sababli, axborot texnologiyalarini yanada rivojlantirish O'zbekiston davlat siyosatining ustuvor yo'nalishlaridan biri bo'lib qolmoqda. So'nggi yillarda axborot texnologiyalari sohasidagi malakali mutahassislarga va ularning tayyorgarlik darajasiga talablar o'sib bormoqda. Bunday mutaxassis dasturiy vositalarga talablarni shakllantirish, ularning sifati va samaradorligini baholash, foydalanuvchi so'rovlariga mos keluvchi dasturiy vositalarni tanlashni bilishi hamda yangi dasturiy mahsulotlarni ishlab chiqish va tayyor dasturiy mahsulotlarni qo'llashning aniq sharoitlariga moslashtirishda ishtirok etishi kerak.

Iqtisodiyotimiz va jamiyatimiz hayotida axborot-kommunikatsiya texnologiyalarining alohida va muhim o'rin tutishini hisobga olib, 2013-yilda 2013–2020-yillarda O'zbekiston Respublikasining Milliy axborot-kommunikatsiya tizimini rivojlantirish kompleks dasturi qabul qilindi."[1] – deb ta'kidlab o'tdilar.

Kompyuter yordamida biror bir masalani yechish talab qilinsa, uning yechimini formallashtirish, ya'ni algoritmni qandaydir formatdagi amallar ketma-ketligiga keltirish zarur. So'ngra, foydalanuvchining kompyuter bilan muloqot qilishi maksimal qulay bo'lishi uchun interfeysni shakllantirish talab qilinadi. Bu masalani yechish loyihalashning eng qiyin qismlaridan biri hisoblanadi.

Faraz qilaylik, qandaydir omborda material qiymatlarning harakatini hosoblash zarur: talab qilingan kunda nimadan qancha, omborga nechta kirib keldi, ombordan nechta chiqib ketdi, qancha qoldi va boshqalar.

---

[1] Karimov I.A. 2015-yilda iqtisodiyotimizda tub tarkibiy o'zgarishlarni amalga oshirish, modernizatsiya va diversifikatsiya jarayonlarini izchil davom ettirish hisobidan xususiy mulk va xususiy tadbirkorlikka keng yo'l ochib berish – ustuvor vazifamizdir. // "Xalq so'zi", 2015-yil, 18-yanvar

Masalani o‘rganib, uni kompyuterda yechish algoritmini qurilib, bo‘lajak foydalanuvchining kompyuter bilan muloqot qilish interfeysi ishlab chiqilganidan so‘ng, uni kompyuter tushunadigan tilga o‘tkazish lozim, ya’ni ma’lum bir ketmaketliklarni ifodalovchi algoritmik tilda yozilgan mashina programasini yozish zarur. Interfeysni hiosobga olgan holda ishlab chiqilgan aniq masalani yechish uchun har qanday algoritmik til ham mos kelavermaydi. Shu sababli, vaziyatdan kelib chiqqan holda ishlanmaga mos keluvchi va ma’lum bir talablarga javob beruvchi tilni tanlash lozim. U nafaqat uning dasturlash qismini, balki bir qancha shartlarga: dasturchiga tez va ishonchli dasturni yaratish imkonini berishi, foydalanish davrida qulay nazoratni ta’minlashi lozim va boshqalar.

O‘zbekiston Respublikasi uzluksiz ta’lim tizimi muassasalarini yangi avlod o‘quv adabiyotlari bilan ta’minlash, talaba-o‘quvchilarni komil inson qilib yetishishiga qaratilgan darsliklar va o‘uv adabiyotlarini yaratish bugungi kunnig dolzarb vazifasidir. Barkamol avlod tarbiyasini va raqobatbardosh kadrlarni tayyorlashni zamonaviy adabiyotlarsiz tasavvur qilish mushkul.

Yangi avlod adabiyotlarini yaratishning me’yoriy-huquqiy asosi bo‘lib O‘zbekiston Respublikasinnig “Ta’lim to‘g‘risida”gi Qonuni va Kadrlar tayyorlash milliy dasturi, Vazirlar Mahkamasining 1998-yil 5-yanvardagi “Uzluksiz ta’lim tizimini darsliklar va o‘quv adabiyotlari bilan ta’minlashni takomillashtirish to‘g‘risida”gi 4-sonli, 2000-yil 29-maydagi “O‘quv darsliklari, darsliklar va o‘quv qo‘llanmalarni qayta ko‘rib chiqish va yangilarini yaratish bo‘yicha Respublika Muvofiqlashtiruvchi komissiyasini tuzish to‘g‘risida”gi 208-sonli qarorlari hisoblanadi.

Ilm-fan jadal taraqqiyot etgan, zamonaviy axborot-kommunikatsiya tizimi vositalari keng joriy etilgan jamiyatda turli fan sohalarida bilimlarning tez yangilanib borishi, ta’lim oluvchilar oldiga ularni jadal egallash bilan bir qatorda, muntazam va mustaqil ravishda bilim izlash vazifasini qo‘ymoqda[2].

[2] Oliy va o‘rta maxsus ta’lim vazirligining 2013-yil 2-avgustdagi 278-sonli buyrug‘iga 1-ilova

Dasturlash haqida umumiy ma'lumotlar

Qandaydir algoritmik tilda dastur yozilganidan so'ng, u mashina tiliga o'tkazilishi lozim. Buning uchun kompilyator deb nomlanuvchi maxsus dasturlar mavjud. Bu dasturlar parametrlarga ega bo'lib, ular kompilyatorga u yoki bu tekislikda mashina dasturlarini yaratish imkonini beradi. Masalan, kompilyatorga jamlangan dastur band qiladigan xotira o'lchamini minimalashtish, boshqariluvchi yoki boshqarilmaydigan kodlardan tashkil topgan dastur (odatda shunchaki «kodlar» deyiladi) yaratish imkonini beruvchi parametrlar. Kompiliyatiorning parametrlari turlicha nomlanadi: kalitlar, opsiyalar.

Aniq masalani yechish uchun (mashina algoritmini shakllantirish) standart dasturlarni (masalan, o'nli sonni ikkilik yoki o'n oltilik songa o'tkazish) olgan kutubxonalarni qo'shish talab qilinadi. Amaliy dastur yozayotgan hech bir dasturchi har doim bunday almashtirish bilan shug'ullanmaydi. Shu sababli, dasturlash muhitini ishlab chiquvchilar yaratilayotgan muhit (bizning holda Visual Studio 2010–13) doirasida bunday kutubxonalarni yaratadilar va kompilyatorlarni o'z ichiga olgan muhit bilan birga taqdim etadilar. O'z navbatida malakali dasturchi mustaqil ravishda bunday kutubxonani yaratishi va uni keyinchalik masalalani yechishga avtomatik ravishda ulash uchun dasturlash muhitining kutubxonalari umumiy ro'yxatiga qo'shib qo'yishi mumkin. Bunday maqsadlar uchun muhit maxsus vositalar taqdim etadi.

Kompilyator kodni ko'rib chiqqan holda uni mashina buyruqlariga o'tkazadi. Bunda nafaqat buyruqlar to'plami, balki alohida modullar (obyekt) ko'rinishidagi to'plamni shakllantiradi. har bir bunday modulda ularning joylashuv joylariga murojaatlarni olgan o'zining nomlar jadvali yaratiladi. Shunay qilib, kompilyator obyekt kodi deb nomlanuvchi noravshan (yechilmagan, ya'ni aniqlanmagan) murojaatlarni oluvchi bajarilmaydigan kod yaratadi. Bajariluvchi modul olish uchun kom-

pilyator tomonidan berilgan kodni u uchun ajratilgan kompyuter xotirasiga aniq joylashtirishni hisobga olgan holda shakllantirilgan murojaatlarni "yechish" (ya'ni aniqlashtirish) lozim. Bu ishni maxsus dastur bajaradi va u turlicha nomlanadi: bog'lanishlar redaktori, kompanovkachi, bog'lovchi, quruvchi. Bu dasturning ishidan so'ng, kompyuterda bajarish uchun tayyor bo'lgan, mashina buyruqlari to'plami shaklanadi.

Dasturiy mahsuhlotlarning krizisi davrida, (60-yillarda) strukturali dasturlash tushunchasi yuzaga keldi. Ya'ni katta avtomatlashgan tizimlarni ishlab chiqish cho'zilib ketdi. Dasturchilar muddatida ishni tugata olmadilar. Tayyor dasturlarda esa ko'plab xatoliklar aniqlandi. 1968-yilda gollandiyalik matematik-dasturchi Deykstra muammoning yuzaga kelishi sabablarini aytib o'tdi: katta dasturlarda aniq mantiqiy struktura mavjud emas, ular *asossiz, juda murakkab*. Bu «asossiz» murakkablikni goto buyrug'i yuzaga keltiradi. Agar dastur ko'plab goto operatoriga ega bo'lsa, insonga boshqaruvning uzatilishini (boshqaruv ketma-ketligi) kuzatib borish juda qiyin bo'ladi, kompyuter bu qiyinchilikni sezmaydi.

Deykstra goto operatorlari o'rniga uch turdagi boshqaruvchi strukturalardan foydalanishni taklif qildi. Birinchi tur – bu oddiy ketma-ketlik (rioya qilish), bunda operatorlar bir biridan so'ng chapdan o'ngga va yuqoridan pastga bajariladi. Ikkinchi tur – bu tarmoqlanish, shart bo'yicha tanlash (if-else), ko'plik tanlov (switch-case). Va nihoyat strukturani boshqaruvchi uchinchi tur – sikl (for), ya'ni bir yoki bir necha operatorning sikldan chiqish sharti bajarilmaguncha takrorlanishi.

IBM firmasi Deykstraning g'oyalaridan, ya'ni strukturali dasturlash prinsiplaridan «Nyu-York times» gazetasining berilganlar bazasini yaratish uchun unumli foydalandilar. Shundan boshlab, strukturali dasturlash konsepsiyasi nazariy va amaliy dasturlash ta'minotining barcha darajalariga sezilarli ta'sir ko'rsatdi.

Bugungi kunda dasturchilar dasturni mantiqan yaxlit bo'lgan qism dasturlar, funksiyalar va sinflar ko'rinishidagi hisoblash bloklariga bo'lishga harakat qilmoqdalar. Bu hisoblash bloklari iyerarxik strukturani ifodalaydi. U yoki bu ma'noviy bloklarni ko'zdan yashirgan holda dasturning murakkabligi berkiladi. Bu esa, dasturning mantiqan tushunarli bo'lishini ta'minlaydi.

Har bir hisoblash bloki imkon qadar ko'proq izohlarga ega bo'lishi lozim (blokning vazifasi, qanday texnologiya va qoidalar qo'llaniladi). Obyektga yo'naltirilgan dasturlash konsepsiyasi – bu dasturni ko'proq strukturalash pog'onalaridan biri. Zamonaviy dasturlash tizimlari dasturlarning maksimal o'qiluvchanligini ta'minlashga harakat qiladi. Chekinishlar, dastur kodining ranglar bilan ajratilishi uning ifodaliligi va tushunarliligini ta'minlaydi.

Shuni hisobga olish lozimki, bir hafta oldin yozilgan dasturning mazmuni tezda unutiladi. Bir oy oldin yozilgan dasturga qarasangiz boshqa dasturchi yozganga o'xshaydi. Hayron bo'lmang, bu inson tabiatining xususiyatlaridan biri – keraksiz narsalarni tashlab yuborish.

Dastur kodini yozish davrida obyektlarga, o'zgaruvchilarga protsedura-larga imkon qadar aniq nom berish lozim, shunda yozilgan operatorlarning mazmuni yanada ochilgan bo'ladi.

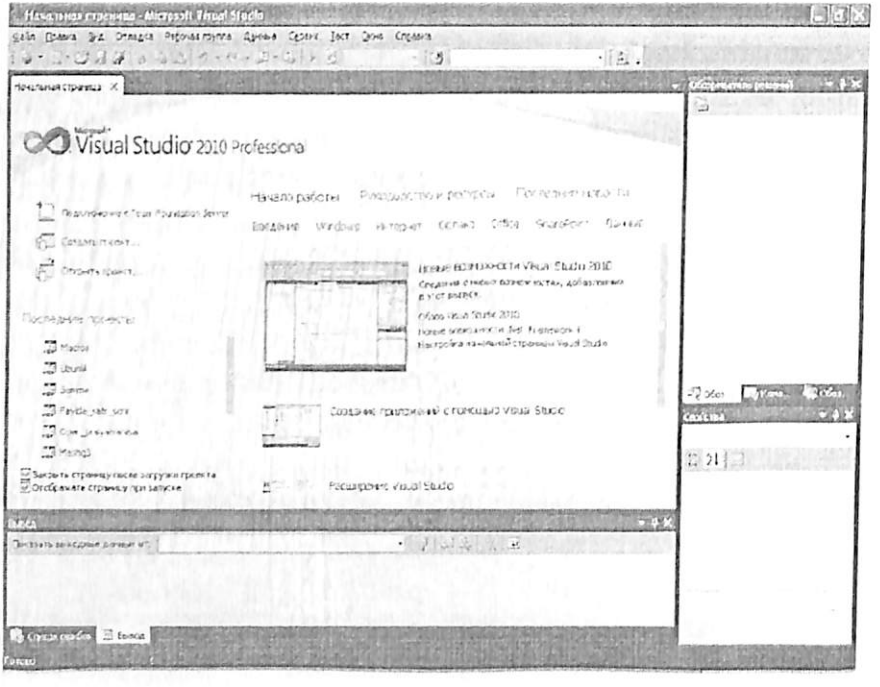
Ushbu qo'llanmadan foydalanishda dasturlash bo'yicha boshlang'ich tushunchalarga ega bo'lish lozim.

1. C++ tiliga kirish

1.1. Konsol ilovasini yaratish

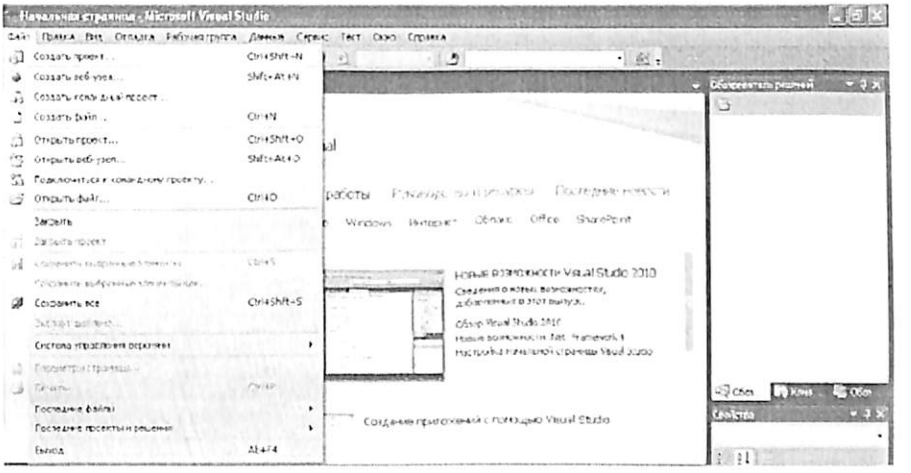
O'rnatilgan dastur mahsulot bilan ishlashda qulaylik uchun uning nishonini operatsion tizimning ishchi stolida joylashgan dasturlarni tez ishga tushirish lentasiga (stolning quyi qismida joylashgan) olib o'tish mumkin. Bu lentadagi har qanday dasturiy mahsulot sichqoncha tugmasini uning nishonida bir marta bosish bilan ishga tushiriladi.

Microsoft Visual Studio dasturlash muhiti (qulaylik uchun VC deb nomlaymiz) kompyuterda o'rnatilganidan so'ng, uni bosh menyuning Пуск|Программы buyrug'idan foydalanib ishga tushirish mumkin. VC muhitini kompyuter ekraniga yuklaymiz. Ekranda bosh oyna hosil bo'ladi.

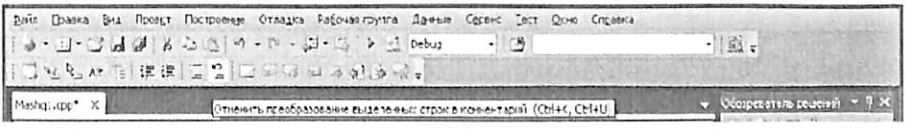


Oynaning yuqori qismida muhitning bosh menyusi buyruqlarini olgan satr joylashgan (Файл (File), Правка (Edit)...) – bu gorizohtal menyu satri. Bu buyruqlar chaqirilganda (ular opsi-

yalar, ya'ni bir necha qiymatlardan birini tanlash elementlari, deb ham nomlanadi) «tushuvchi menyu» deb nomlanuvchi menyu ochiladi – bu yuqoridan quyiga qarab joylashgan buyruqlar to'plamini aks ettiruvchi vertikal menyu.



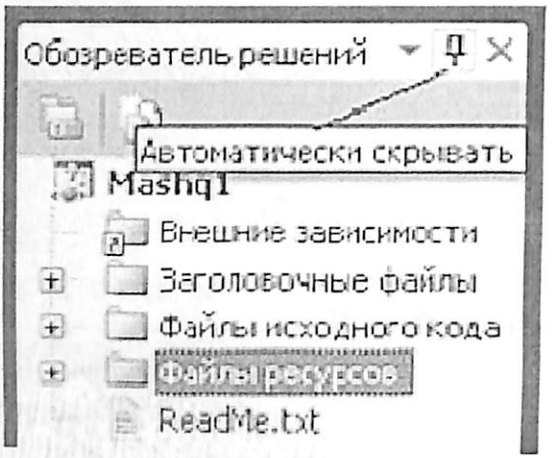
Bosh menyular satridan so'ng ba'zi buyruqlarni bajarilish uchun tez chaqirish tugmalari joylashgan. Bu tugmalarning barchasi qalqib chiquvchi yordamga ega (sichqoncha ko'rsatkichi olib kelinib ozgina kutilsa, namoyon bo'ladi). Bunday tugmalar yonida asosiy tugmaning qiymatlari ro'yxatini ochuvchi qo'shimcha tugma bo'lishi mumkin. Barcha tugmalar ishchi stolda ularga ajratilgan joyga sig'masa, huddi, Worddagi kabi, ular ochish tugmasini olgan yo'lakchali tugmaga joylashtiriladi.



O'z navbatida bosh oynaning ko'rinishi yaratilayotgan ilovani berish jarayonida o'zgaradi. Ishchi stol oynalar to'plamidan shakllanadi. har bir oyna – bu o'zining yuqori qismida standart sarlavha yo'lakchasiga ega bo'lgan oddiy windows-oyna. Bu

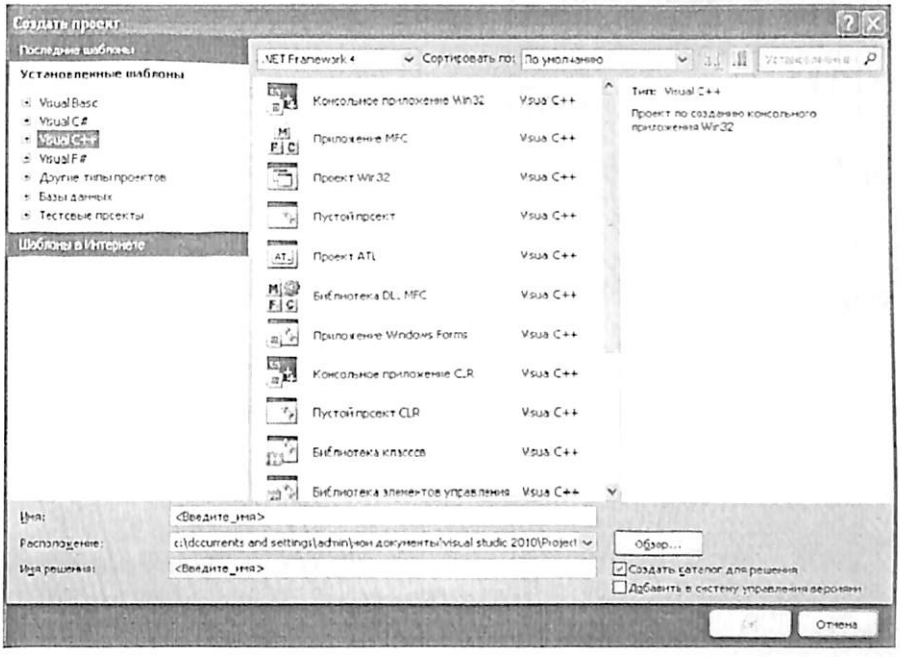
yo‘lakcha yordamida oynaning ekranda joylashuvini o‘zgartirish mumkin. Oynalar xossalarga ega bo‘lib, ularni ko‘rish uchun sarlavhada sichqoncha o‘ng tugmasi bosiladi.

Oynalarning ishchi stolda joylashuvini boshqarish uchun ularning xossalarini yaxshi bnilish lozim. Ishchi stolda oynalarning ko‘pligi ishga halaqt beradi. Bunday holarda ba’zi oynalarni berkitib qo‘yish yoki asosiy oynaning yon tomonlariga joylashtirish lozim.

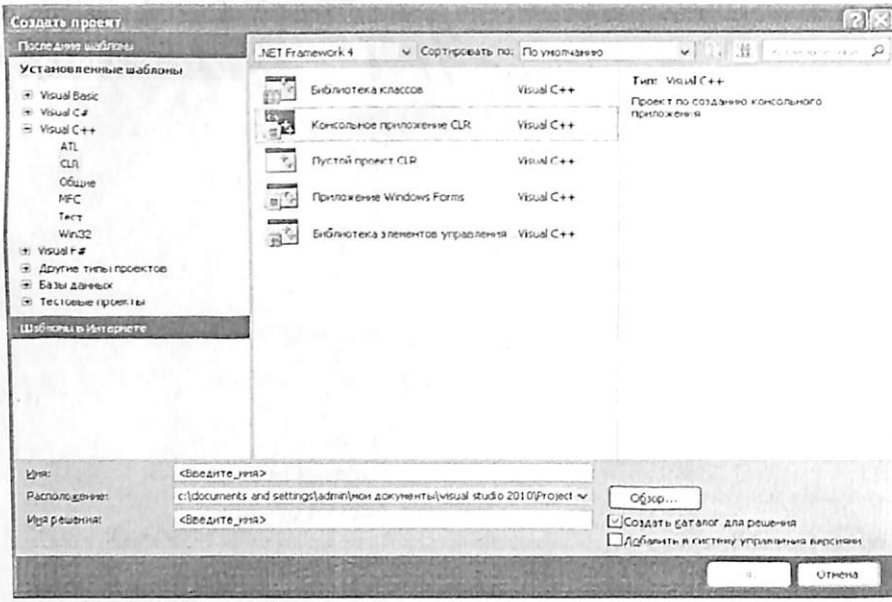


Oynalarni avtomatik yopib qo‘yish uchun Auto Hide (*avtomatik berkitish*) tugmasidan foydalaniladi. Tugma vertikal yarim o‘tkazgich ko‘rinishiga ega. Agar unda sichqonchaning chap tugmasi bosilsa, muhit bosh oynasining o‘ng tomoniga joylashadi va berkitiladi (faqat uning nomi ko‘rinib turadi). Sichqoncha ko‘rsatkichi uning nomiga olib kelinsa, u avtomatik ochiladi va uning sarlavha qismida gorizohtal yarim o‘tkazgich ko‘rinishidagi Auto Hide nishonini ko‘ramiz.

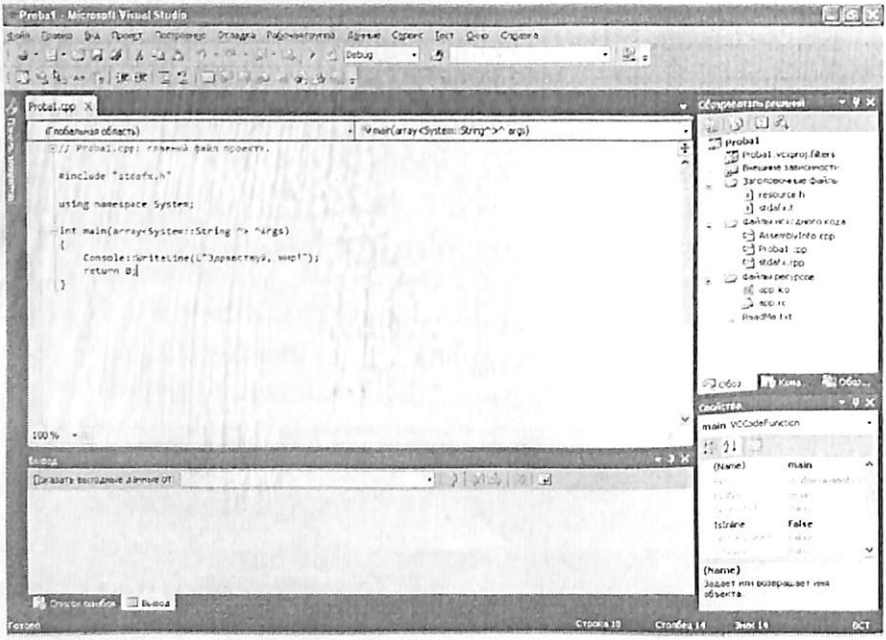
CLR-ilova qolipida konsol ilovasini yaratish uchun Visual Studio muhiti yuklanadi. Bosh menyuning Файл=>Создать проект buyrug‘i bajariladi. Natijada rasmda ko‘rsatilganidek muloqot oynasi ochiladi.



Bu oynadan Visual C++ qo‘yilmasi tanlanadi. Ochilgan ro‘yxatdan CLR satri, so‘ngra buyruqlar ro‘yxatidan Консолное приложение *CLR* (CLR konsol ilovasi) opsiyasi tanlanadi. Uning quyi qismida loyihaning nomi Имя (Nom), yechim nomi Имя решения (Yechim nomi) va loyiha joylashtirilishi lozim bo‘lgan papka (Обзор (Ko‘rish) orqali) ko‘rsatiladi.



OK tugmasi bosilganidan so‘ng quyidagicha oyna hosil bo‘ladi:



Tayyorlama bosh funksiyasi sarlavhadan

```
// Proba1.cpp: главный файл проекта.
#include "stdafx.h"
using namespace System;
int main(array<System::String ^> ^args)
```

va figurali qavslar bilan chegaralangan tanadan tashkil topgan.

```
{
 Console::WriteLine(L"Здравствуй, мир!");
 return 0;
}
```

main(argumentlar to'plami) funksiyasining sarlavhasini main() ko'rinishiga almashtiramiz, ya'ni argumentsiz holatga keltiramiz. Tana qismidan return 0 operatorini o'chiramiz. Buning hammasini konsol ilovasi tayyorlamasi ekranda hosil bo'lishi bilan birga paydo bo'luvchi Kod redaktori orqali bajaramiz (Tayyorlama Kod redaktori maydoniga joylashadi). Bu bosh funksiya bir necha konsol ilovalarini bog'lash uchun xizmat qiluvchi argumentlarga ega emasligini bildiradi.

Konsol ilovasining tayyorlamasi shaklantirilganda va ilovaga nom Имя (Nom) berilganda Решение (Yechim) maydoni shakllanadi. VC++ muhiti yaratilayotgan ilovani birining ichiga ikkinchisi joylashtirilgan ikkita konteyner ko'rinishida jixozlaydi. Biri (bosh konteyner) Решение (Yechim) deb nomlanadi, boshqasi esa – Проект (Loyiha). Loyiha fayllar gurugini birlashtiruvchi konfiguratsiya (karkas, konteyner) sifatida aniqlangan.

Loyiha doirasida bajarilishi lozim bo'lgan, ya'ni kompilirovka qilingan va doimiy, dastur shakllantiriladi. har bir loyiha eng kamida ikkita konfiguratsiyaga ega: sozlovchi va oddiy (bajariluvchi). Bu (tushuvchi) menyuda beriladi va ko'rsatilmaganda Отладка (Sozlash) opsiyasiga o'rnatilgan. Tushuvchi menyu muhit oynasi bosh menyusi ostidagi satrda joylashgan.

Loyihalar Решение (Yechim) deb nomlanuvchi boshqa karkasning, konteynerning qismi hisoblanadi va loyihalar

o‘rtasidagi aloqalarni tasvirlaydi: bitta Yechim ko‘plab loyihalarni, loyiha mavjudligini ta’minlovchi ko‘plab elementlarni olishi mumkin. Yechim – bu birlashgan loyihalar to‘plami. Uni shunchaki, “Yechim” termini bilan almashtirmaslik uchun “Loyihalar guruhi” deb ataymiz. Loyihalar guruhi bilan ishlash uchun, Solution Explorer deb nomlanuvchi, maxsus anjom mavjud. Unga ishlab chiqish muhitining Просмотр (Ko‘rish) opsiyasi orqali o‘tish mumkin. Muhitning o‘zi yaratilayotgan ilovani loyihalar guruhi sifatida avtomatik shakllantiradi.

Ilovani jixozlashga bunday yondoshuv loyihalar guruhi bilan ishlash imkonini yaratadi. Bu esa o‘z navbatida ilovalarni ishlab chiqish jarayonini tezlashtiradi.

main() funksiyasi tanasida quyidagicha ikkita satrni yozamiz:

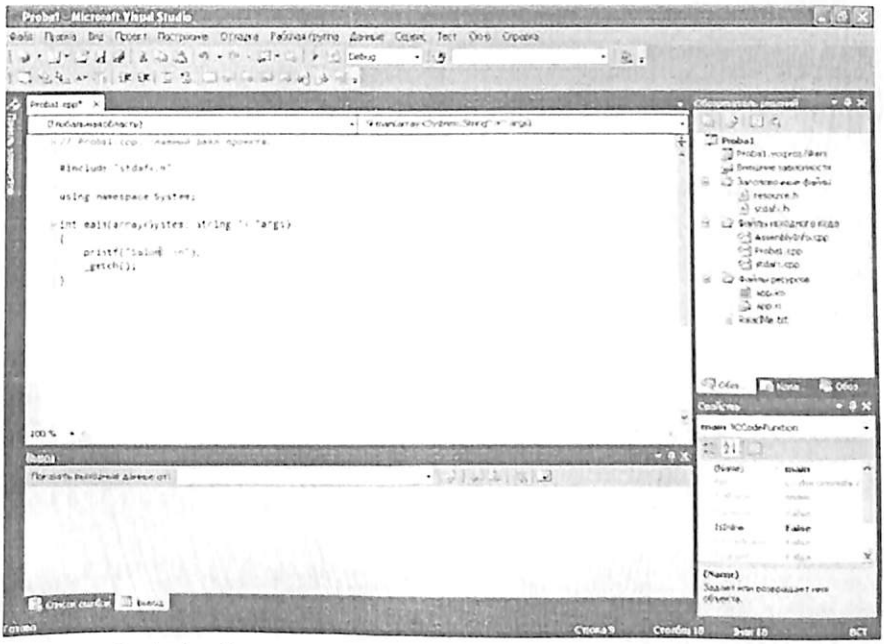
```
printf (“Salom! \n”);
_getch();
```

Bu ilovaning birinchi kodi. U ekranga “Salom!” matnini chiqarishi va tasvirni o‘chib ketmasligi uchun ushlab turishi lozim.

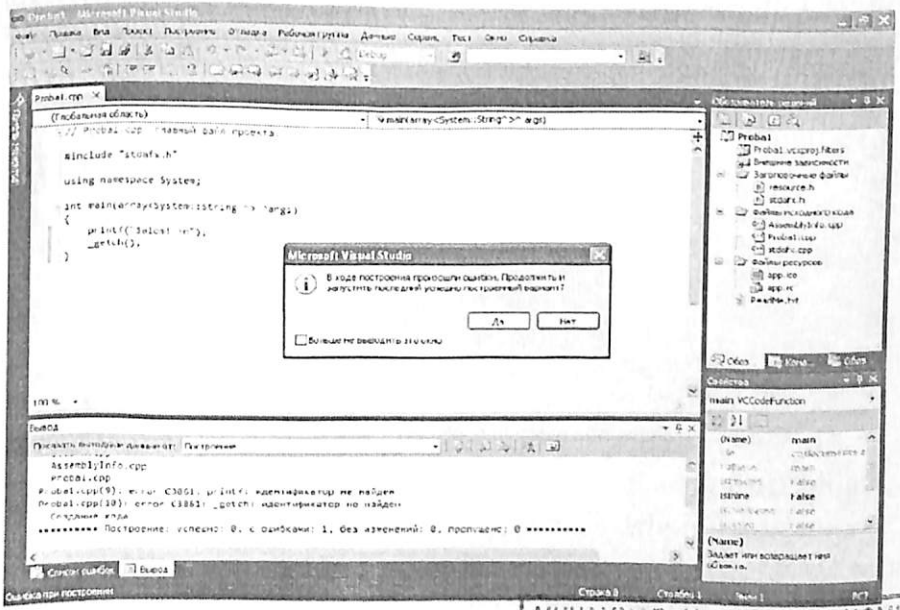
printf(“Salom! \n”); operatori matni ekranga chiqarishni, ekrandagi ma’lumotlarni ushlab turishni _getch(); operatori ta’minlaydi.

E’tibor bergan bo‘lsangiz, nuqtali vergul bilan tugalangan qandaydir ifoda operator deb ataladi. Birinchi operatorga printf(“Salom!\n”) funksiyasi, ikkinchi operatorga _getch() funksiyasi kiradi. Natijada konsol ilovasi rasmda ko‘rsatilganidek ko‘rinish oladi.

Ilova ishlashi uchun uni kompilyatsiya qilish, ya’ni VC++ tilida yozilgan ma’lumotlarni mashina kodiga o‘tkazish zarur. Kompilyatsiya qilish uchun [Ctrl]+[Shift]+[B] klavishlar kombinatsiyasi bosiladi yoki bosh menyudan Построение/Решение (Qurish/Yechim) opsiyasini bajarish orqali kompilyator-dastur ishga tushuriladi.



Amal bajarilgandan keyin qiyudagi rasmda keltirilgan ko‘rinishni olamiz.



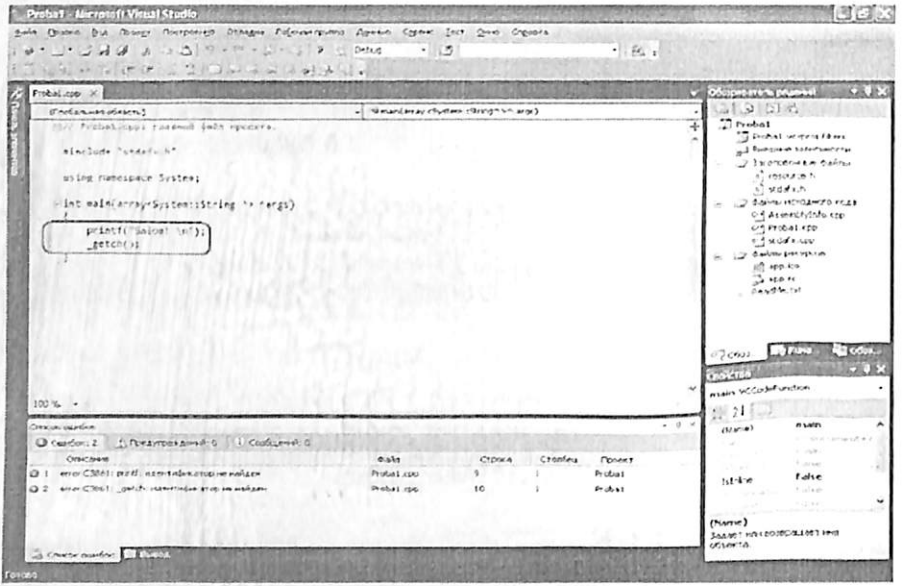
Rasmdan ko'rinib turibdi-ki, kompilyatsiya jarayoni samarali amalga oshmadi: chiqarish oynasida xatoliklar haqida xabar chiqarildi. Xususan quyidagi xabarlarni ko'ramiz:

"error C3861: 'printf': идентификатор не найден".

«error C3861: '_getch': идентификатор не найден».

Bu shuni anglatadiki, kompilyator printf va _getch funksiyalarini bajarish uchun hech narsa topmadi (ya'ni ularni "tanimadi").

Agarda xatolik haqidagi xabarni olgan har bir satrda sichqoncha tugmasi ikki marta bosilsa, main() funksiyasi chap tomonidagi maydonda xatolik topilgan satrlar belgilanadi. Aniqlangan xatoliklar ustida ishlaymiz. Dastur sarlavhasida quyidagini yozamiz: #include <conio.h>

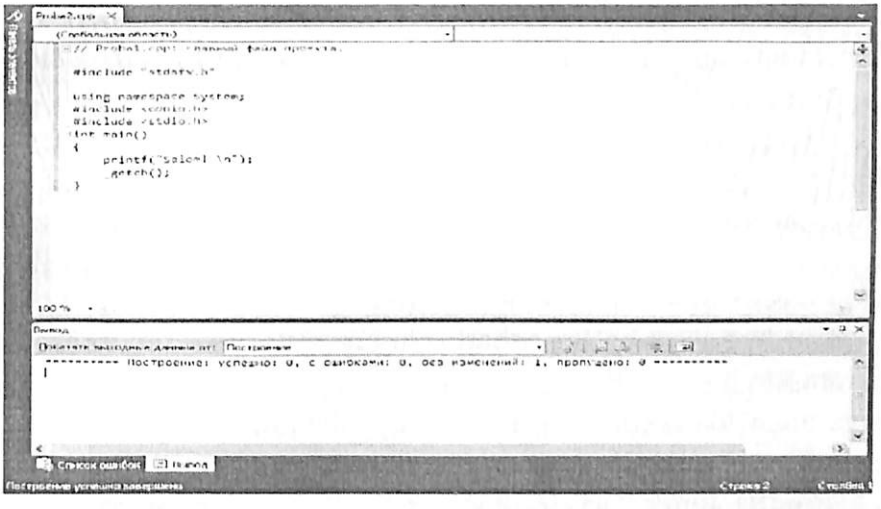


Bu fayl yaratilayotgan dasturga ulansa, funksiya haqidagi xatoliklarni topadi va xatolik to'g'rilanadi. Funksiyaning bayonini olgan faylni qo'shish (Header file – bu mundarija fayli, kengaytmadagi "h" belgisi shu yerdan olingan) #inciude operatori orqali amalga oshiriladi. Bu kompilyator operatori. U das-

tur moduli matniga “<”, “>” belgilari ichida ko‘rsatilgan faylni qo‘shadi. Bunday fayllarda dastur uchun zarur axborotlarni o‘zingiz berishingiz ham mumkin. Bunda dasturga qo‘shish zarur bo‘lgan fayl nomi #include operatorida qo‘shtirnoq ichida ko‘rsatiladi (#include “conio.h”).

Kompilyatsiyaning ikkinchi xatoligi ustida ham huddi shunday yo‘l tutamiz. Kompilyator printf() funksiyasini “tanishi” uchun dasturga #include <stdio.h> sarlavha fayli qo‘shiladi.

[Ctrl]+[F5] klavishlar kombinatsiyasi orqali kompilyatorni ishga tushiramiz. Dastur matni redaktori ikkita og‘ma chiziqni (//) izohning boshlanishi – dasturning bajarilishida ishtirok etmaydigan belgilar to‘plami sifatida ko‘radi. Aslida kompilyatsiya alohida ishga tushirilmaydi, balki Откомпилировать и построить (*Kompilyatsiya qilish va qurish*) rejimi bajariladi. Bunday rejimdan dasturni birinchi bor ishga tushirganda foydalandik. Kompilyatsiya natijasi quyidagi rasmda keltirilgan.



Rasmdan ko‘rinib turibdiki, xatoliklar yo‘qoldi. Endi dasturni bajarilishi uchun ishga tushirish mumkin. Buning uchun запустить без работы с Отладчиком (*Sozlagich bilan ishlamasdan ishga tushirish*) buyrug‘ini bajarish yoki [Ctrl]+[F5]

klavishlar kombinatsiyasini bosish talab qilinadi, yoki qurishdan keyin exe-modul (loyihani qurish dasturi ishining natijasi) shakllangan katalogga kiriladi va uning nishonida sichqoncha tugmasi ikki marta bosiladi. Agar dastur bajarilsa, qora rangli oynada "Salom!" matni chiqariladi. So'ngra ihtiyoriy klavish bosilganda dastur ishini tugallaydi va yana dastur matniga qaytiladi. Yangi loyihani saqlash uchun File=>Save All (Файл=>Сохранить всё) buyrug'i bajariladi.

Konsol dasturining bajarilishi natijasi

Har qanday C++-dastur funksiya deb nomlanuvchi elementlar to'plamidan quriladi – ma'lum bir amalni bajaruvchi dastur kodlari bloki. Maxsus qoidalar bo'yicha qurilgan bu blok-kodlarning nomlarini dasturchi beradi yoki dasturlash muhiti tomonidan qo'yilgan standart funksiyalar kutubxonasida beriladi. Ilovaning bajarilishi boshlanadigan bosh (asosiy) funksiyaning nomi (main()) muhit tomonidan beriladi.

Dasturning bajarilishi jarayonida main() funksiyasi boshqa funksiyalar bilan ma'lumot almashadi va ularning natijalaridan foydalanadi. Funksiyalar o'rtasida ma'lumot almashinish qavs ichida keltirilgan funksiya parametrlari orqali amalga oshiriladi. Funksiya parametrga ega bo'lmasligi ham mumkin, ammo qavslar har doim bo'lishi shart – kompilyator ular orqali funksiyani tanib oladi. Birinchi keltirib o'tilgan misolda bosh funksiya main()da ikkita funksiyadan foydalanildi: bu funksiyalar – printf() va _getch().

printf() funksiyasi argument sifatida belgilar satrini olgan (belgilar qo'shtirnoq ichiga olingan). Bu satrning belgilari ichida quyidagicha yozilgan maxsus belgi bor: '\n'. Bu boshqaruvchi belgi – belgilarni ASCII kodlash jadvalining birinchi 32 ta belgilaridan biri.

Boshqaruvchi belgilar ekran tasviriga ega emas va jarayonlarni boshqarish uchun foydalaniladi. Bizning holatda '\n' belgisi printf() satrning boshqa belgilari bo'lgan funksiya buferini ekranga tasvirlash va ekranda tasvirlarni ifodalash ko'rsatkichini navbatdagi satrning birinchi pozitsiyasiga o'rnatish uchun xizmat qiladi. printf() funksiyasi ishlayotganda satrning belgilari qandaydir buferga '\n' belgisi uchramaguncha bittadan yoziladi. Belgi o'qilganidan so'ng, buferdagi ma'lumotlar chiqarish qurilmasiga uzatiladi, bizning holda bu konsol ilovasi oynasi – ekran.

Konsol interfeysini (nografik) yaratishda har doim ikkita fayl bog'lanadi, ulardan biri – ma'lumotlarni klaviaturadan kiritish uchun, ikkinchisi – ma'lumotlarni ekranga chiqarish uchun.

Agar _getch() funksiyasi olib tashlansa, natijada hech qanday ma'lumot ko'rmaymiz. Funksiya mavjud bo'lganda dastur biror bir belgi kiritilishini kutib turadi. Belgi (masalan, a) kiritilganidan so'ng, funksiya ishini tugallaydi. U bilan birgalikda butun dastur ham o'z ishini tugallaydi, chunki, bu funksiya oxirgi bajariluvchi operator edi.

getch() – klaviaturadan bir belgini kiritishi funksiyasi: u biror klavishni bosishni kutib turadi. Klavish bosilmaguncha dasturning bajarilishi natijasini ekranda ushlab turiladi. Boshlovchi dasturchi ekran "qochib ketishi"ni bunday ushlab turish haqida bilishi lozim. getch() funksiyasining asosiy tayinlanishi belgilarni klaviaturadan kiritish va ularni belgili o'zgaruvchilarga uzatish. Bu yerda biz funksiyaning ikkilamchi xossasidan foydalandik – klaviaturadan kiritishni kutish va shu orqali natijani ko'rmaguncha dasturga ishni tugalash imkonini bermaslik.

1.2. Dasturning tuzilishi va shakli

VC++da dasturlar ***ilovalar*** deb nomlanadi. Ilovalar muhitda foydalanuvchi uchun bir necha fayllar mosligi kabi ko'rinishidagi maxsus konstruksiyalar – loyihalar ko'rinishida quriladi.

VC++ tilidagi dastur – bu funksiyalar mosligidir, ya'ni ma'lum bir talablarga javob beruvchi maxsus dasturlar tuzil-

masi. Bunda ilova bosh funksiya bo'lib, uning ichida ilova algoritmini amalga oshiruvchi operatorlar joylashtiriladi. Operatorlar ichida shudaylari borki, ular algoritmning bajarilishida talab qilinuvchi boshqa funksiyalarni chaqirish uchun xizmat qiladi. Dasturni ishga tushirish, dasturning barcha qismini o'z ichiga olgan, bosh funksiyani ishga tushirishdan boshlanadi. Funksiyalarning bir qismi dasturchi tominidan yaratuiladi, qolgan qismi (kutubxona funksiyalari) dasturlash muhiti tomonidan foydalanuvchiga taqdim etiladi va dasturlarni ishlab chiqish jarayonida foydalaniladi. Dasturlashni o'rganish jarayonida ilovaning maxsus ko'rinishi – konsol ilovasidan foydalanamiz. Bunda dastur kodlari avvaldan tayyorlangan shablonlar asosida shakllantiriladi.

Konsol ilovasi (*tayanch, o'zak*) – bu grafik interfeyssiz ilova bo'lib, foydalanuvchi bilan maxsus buyruq satri yoki (agar ular IDE doirasida ishlayotgan bo'lsa) muhitning bosh menyusidan maxsus buyruq orqali ishga tushiriladi. Bunday ilovalar Файл=>Создать проект=>Новый (Fayl=> Loyiha yaratilsin=>Yangi) buyrug'i bajarilganidan so'ng ochiluvchi maxsus qolip yordamida yaratiladi. Konsol ilovasi qolipi yaratilayotgan ilovaga zarur elementlarni qo'shadi (bo'lajak ilovaning tayyorlamasi yaratiladi), so'ngra ishlab chiquvchi C++ tilidagi o'zining operatorlarini bu qolipga qo'yadi. So'ngra, ilova avtonom bajariluvchi faylga jamlanadi (kompilyatsiya qilinadi) va bajarilish uchun ishga tushirilishi mumkin. Foydalanuvchi bilan muloqot ilova ishga tushirilgandan so'ng ochiluvchi maxsus, konsol oynasi deb ataluvchi, oyna orqali amalga oshiriladi. Bu oynaga dastur xabarlari chiqariladi, shu oyna orqali hisoblashlar uchun ma'lumotlar kiritiladi va hisoblashlar natijalari chiqariladi. Loyihani kompilyatsiya qilish va yig'ish muhit bosh menyo'sining Построение (Qurish) buyrug'i orqali amalga oshiriladi. Kompilyatsiya va yig'ishdan so'ng, loyihani bajarish uchun ishga tushirish mumkin. Bajarilishni ishga tushirish muhit bosh oynasining Debbug opsiyasi yordamida amalga oshiriladi.

Visual C++ da konsol ilovalarini shakllantirishda ikkita qolip berilgan:

- ✓ CLR qolipi – CLR papkasida;
- ✓ oddiy konsol ilovasi uchun qolip – Win32 papkasida.

VC++ muhitida dasturlashni o'rganishda Консолное приложение *CLR* (CLR konsol ilovasi) qolipidan foydalanamiz.

CLR – bu maxsus muhit bo'lib, dastur kodining bajarilishini, xotira, berilganlar oqimi va masofaviy uzoqlashgan kompyuterlar bilan ishlashni boshqaradi. Bunda kod bajarilishining xavfsizligi va ishonchliligi qat'iy ta'minlanadi. CLR Microsoft firmasi tomonidan VC++ 2005 talqinidan boshlab kiritilgan C++ ning qo'shimcha kengaytmasi hisoblanadi.

Eski talqinlarda obyekt bilan ishlaganda ularni muhit tomonidan ajratilgan xotiraga joylash lozim. Ilova uchun "**to'plam**" (куча) deb ataluvchi xotira ajratiladi: unda obyektlar joylashtiriladi. Obyekt bilan ishlash tugallanganda xotirani o'zingiz bo'shatasiz. Aks holda to'plam to'lishi va ilovaning bajarilishi to'xtab qolishi mumkin. Bu boshqarilmaydigan to'plam hisoblanadi.

CLR rejimi yoqilganda ilova odatiydan farqlanadi, uning tayyorlamasi ilovaga, xotirada avtomatik sozlanishi zarur bo'lgan obyektlarni olgan, maxsus System tizim sohasini ulashni ta'minlaydi. CLR rejimi xotiraning boshqariluvchi to'plami bilan ishlaydi. Bunda obyektlarni joylashtirish va ularni bo'shatish muhit boshqaruvi orqali amalga oshiriladi.

Boshqariluvchi ko'rsatkich – bu ko'rsatkich turi bo'lib, xotiraning umumiy boshqariluvchi to'plamida joylashgan uning bajarilishi davrida ilovaga taqdim etilgan obyektlarga murojaat qilaldi (obyektlarga murojjat qilish mumkin bo'lgan xotira manzillari). Bunday ko'rsatkichlar uchun maxsus belgilash: "*" belgisi o'rniga "^" belgisi qo'llaniladi.

Dastur matnli faylda joylashgan ko'rsatmalardan tashkil topgan. Oddiy dasturning tarkibi quyidagicha ko'rinishda bo'ladi:

```
<Sarlavha fayllarini ulash>
<Global o'zgaruvchilarni e'lon qilish>
<Funksiya, sinf va boshqalarni e'lon qilish>
int main([int argc, char *argv[]])
{
<Ko'rsatmalar>
return 0;
}
<Funksiyalar va sinflarning aniqlashi>
```

Dasturning boshida identifikatorlarni nomlashsiz e'lon qilishni olgan sarlavha fayllari qo'shiladi. Masalan, berilganlarni konsol oynasiga chiqarish uchun cout obyektini e'lon qilishni olgan iostream faylini ulash zarur. Fayllarni ulash include direktivasi orqali amalga oshiriladi.

```
#include <iostream>
```

include ko'rsatmasi yordamida ham standart sarlavha fayllarini, ham foydalanuvchi fayllarini ulash mumkin.

Fayllar ulanganidan so'ng *global o'zgaruvchilar* e'loni amalga oshiriladi. Bunday o'zgaruvchilar ma'lum bir tipni saqlash uchun mo'ljallangan. Global o'zgaruvchilar butun dasturda ko'rinadi (funksiyalarda ham). Agar o'zgaruvchi funksiya ichida e'lon qilinsa, u holda o'zgaruvchining ko'rinish sohasi shu funksiya bilan chegaralanadi. Dasturning boshqa qismlarida o'zgaruvchidan foydalanish mumkin emas. Bunday o'zgaruvchilar *lokal o'zgaruvchilar* deyiladi.

Butun tipli o'zgaruvchini quyidagicha e'lon qilish mumkin:

```
int x;
```

O'zgaruvchining oldidagi int kalit so'zi o'zgaruvchi butun sonni saqlash uchun tayinlanganligini bildiradi. O'zgaruvchi e'lon qilinganida unga qiymat berish, ya'ni nomlash ham mumkin. Nomlash uchun o'zgaruvchi nomidan keyin "=" operatori, so'ngra qiymat ko'rsatiladi. Masalan: int x = 10;

E'lon qilish davrida global o'zgaruvchiga qiymat o'zlashtirilmasa, u nol (0) qiymat qabul qiladi. Lokal

o'zgaruvchiga qiymat o'zlashtirilmasa, o'zgaruvchi ihtiyoriy qiymat qabul qiladi. Bunday hollarda o'zgaruvchi "**musor**" (yaroqsiz ma'lumot) oladi deyiladi.

O'zgaruvchilarni nomlashda operatoridan oldin va keyin bo'shliq qo'yish (ihtiyoriy sonda) yoki qo'ymaslik ham mumkin. Bundan tashqari, bo'shliqlar o'rniga tabulyatsiya yoki yangi satrga o'tish belgisidan ham foydalanish mumkin. Masalan, quyidagicha ko'rsatmalar o'rinli bo'ladi:

```
int x = 21;
int y = 85;
int z
=
56;
```

Qiymat o'zlashtirish funksiya e'lonida amalga oshirilayotgan bo'lsa, bo'shliqlarni ko'rsatmaslik qabul qilingan. Bular qat'iy qoidalar emas.

Ko'rsatmaning oxiri satr oxiri emas, balki, nuqtali vergul hisoblanadi. Bir satrda bir necha ko'rsatmalar bo'lishi mumkin. Masalan,

```
int x = 21; int y = 85; int z = 56;
```

Bu qoidada istesno mavjud. Pereprosessor direktivalaridan keyin nuqtali vergul qo'yilmaydi. Bunda ko'rsatma oxiri satr oxiri hisoblanadi. Pereprosessor direktivalarini nomdan oldin turgan "#" belgisi bo'yicha bilib olish mumkin. Pereprosessor direktivasiga tipik misol sifatida sarlavha fayllarini ulashda foydalaniladigan include direktivasini ko'rsatish mumkin.

```
#include <iostream>
```

Global o'zgaruvchilar e'lon qilinganidan so'ng, funksiya va sinflar e'lonlari joylashishi mumkin. Bunday e'lon qilishlar *prototiplar* deyiladi. Funksiya prototipi sxemasi quyidagicha ko'rinishga ega:

```
<Qaytaruvchi qiymat tipi> <Funksiya nomi>([<Tip>
[<Parametr_1>] [, ..., <Tip> [<Parametr_N>]]]);
```

Funksiya e'lonida parametrlar nomlarini ko'rsatimaslik ham mumkin. Berilganlar tipi haqida ma'lumot ko'rsatish yetarli.

Masalan, ikkita butun sonni jamlovchi va ularning yig'indisini qaytaruvchi funksiyaning prototipi quyidagicha bo'ladi:

```
int summa(int x, int y);
int summa(int, int);
```

Funksiya e'lon qilinganidan so'ng, funksiyaning aniqlanishi deb nomlanuvchi, amalga oshirishni bayon qilinishi lozim. Funksiyaning aniqlanishi odatda main() funksiyasidan keyin joylashtiriladi. summa() funksiyasining aniqlanishiga misol:

```
int summa(int x, int y) {
 return x + y;
}
```

Misolda summa() funksiyasining aniqlanishi funksiya e'loni bilan mos keladi. Funksiya e'lon qilinganidan so'ng nuqtali vergul qo'yiladi. Funksiyaning aniqlanishida figurali qavslar ichida funksiyaning bajarilishi bayoni bo'lishi lozim. Funksiya e'lonida butun qiymat qaytarilishi ko'rsatilganligi sababli, bayondan keyin qiymat qaytarish zarur. Qaytariluvchi qiymat return kalit so'zidan keyin ko'rsatiladi.

Standartga ko'ra main() funksiyasi ichida return kalit so'zini ko'rsatmaslik ham mumkin. Bunda kompilyator "0" qiymat qaytaruvchi ko'rsatmani o'zi mustaqil ravishda qo'yadi. Qaytariluvchi qiymat operatsion tizimga uzatiladi va dasturning to'g'ri yakunlanganligini aniqlash uchun foydalanilishi mumkin. Agar funksiya qiymat qaytarmasa, funksiya nomidan oldin berilganlar tipi o'rniga void kalit so'zi ko'rsatiladi.

Qiymat qaytarmaydigan funksiyaga misol: void print(int);

print() funksiyasining aniqlanishiga misol:

```
void print(int x) {
 std::cout << x << std::endl;
}
```

Figurali qavslar ichidagi ko'rsatmalardan oldin bir xil cheki-

nish qoldirish lozim. Chekinish sifatida bo'shliqlar yoki tabulyatsiya belgisi qo'yish mumkin. Bo'shliqlardan foydalanilganda blok uchun chekinish uch yoki to'rt bo'shliqdan iborat bo'ladi. Bir dasturda chekinish sifatida ham bo'shliq, ham tabulyatsiyadan foydalanilmaydi. Ulardan birortasini tanlash va undan foydalanish lozim. Yopuvchi fugurali qavs blokning oxirida odatda alohida satrda joylashadi. Qavs oxirida nuqtali vergul qo'yilmaydi.

Dasturda eng asosiy funksiya main() funksiyasi hisoblanadi. Aynan bu funksiya dasturni ishga tushirishda avtomatik ravishda chaqiriladi.

Funksiya ikkita prototipga ega:

int main(); va int main(int argc, char *argv[]);

Funksiyaning birinchi prototipi sinonimga ega: int main(void);

void qiymati funksiya qiymat qabul qilmasligini bildiradi. Bunday sintaksisdan C tilida dasturlashda foydalaniladi. C++ tilida void kalit so'zini ko'rsatish ortiqcha hisoblanadi. Bo'sh qavslarni ko'rsatish yetarli.

Ikkinchi prototip dasturni ishga tushirishda buyruqlar satridan qiymatlarni olish uchun qo'llaniladi. Qiymatlar soni argc parametri orqali, qiymatlarning o'zi esa argv parametri orqali ochiq.

Quyida yuqorida ko'rib o'tilgan struktura uchun dastur kodi keltirilgan.

```
#include <iostream> // Sarlavha fayllarini ulash
int x = 21; int y = 85; // Global o'zgaruvchilar e'loni
// ---Funksiya, sinf va boshqalarni e'lon qilish---
int summa(int x, int y);
// ------------ Asosiy funksiya ------------
int main() {
 int z; // Lokal o'zgaruvchini e'lon qilish
 z = summa(x, y); // summa()funksiyasini chaqirish
```

```
std::cout << z << std::endl;
system("pause");
return 0;
}
// -------- Funksiyaning aniqlanishi (bayoni) --------
int summa(int x, int y) {
return x + y;
}
```

```
106
Для продолжения нажмите любую клавишу . . .
```

1.3. Berilganlar turlari

C++ tilida quyidagi elementar berilganlar tuplari aniqlangan:

bool – mantiqiy tir berilganlar. true ('1' soniga mos keladi) yoki false ('0' soniga mos keladi) qiymat olishi mumkin. O'zgaruvchini e'lon qilishga misol: bool is_int;

char – belgi kodi. Qiymatlar oralig'i: -128 dan 127 gacha. O'zgaruvchini e'lon qilishga misol: char ch;

int – ishorali butun son. Qiymatlari oralig'i operatsion tizimning razryadiga bog'liq. 16 bitli operatsion tizimda oraliq -32768 dan 32767 gacha. 32 bitli operatsion tizimda oraliq -2147483648 dan 2147483647 gacha. O'zgaruvchini e'lon qilishga misol: int x;

float – haqiqiy son. O'zgaruvchini e'lon qilishga misol: float y;

double – ikkilangan aniqlikdagi haqiqiy son. O'zgaruvchini e'lon qilishga misol: double z;

Berilganlarning elementar turidan oldin quyidagi modifikatorlar yoki ularning kombinatsiyasi ko'rsatilishi mumkin:

signed – belgili yoki butun tiplar manfiy qiymat olishi mumkinligini ko'rsatadi. signed char tipi char tipiga mos keladi, -128 dan 127 gacha qiymat qabul qilishi mimkin. O'zgaruvchini e'lon qilish: signed char ch;

signed int tipi int tipiga mos keladi. O'zgaruvchini e'lon qilishga misol: signed int x;

unsigned – belgili yoki butun tiplar manfiy qiymatlar olishi mumkin emasligini ko'rsatadi. unsigned char tipi 0 dan 255 gacha qiymat qabul qilishi mumkin. O'zgaruvchini e'lon qilishga misol: unsigned char ch;

unsigned int tipining qiymatlari oralig'i operatsion tizimning razryadiga bog'liq. 16 bitli operatsion tizimda oraliq 0 dan 65535 gacha. 32 bitli operatsion tizimda oraliq 0 dan 4294967295 gacha. O'zgaruvchini e'lon qilishga misol: unsigned int x;

short – butun tipdan oldin ko'rsatilishi mumkin. short int va signed short int tiplarining qiymatlari oralig'i -32768 dan 32767 gacha. O'zgaruvchini e'lon qilishga misol: short int x; signed short int y;

unsigned short int tipining qiymatlari oralig'i 0 dan 65535 gacha. O'zgaruvchini e'lon qilishga misol: unsigned short int x;

long – butun tipdan va double tipidan oldin ko'rsatilishi mumkin. long int va signed long int tipining qiymatlari oralig'i -2147483648 dan 2147483647 gacha. O'zgaruvchilarni e'lon qilishga misol: long int x; signed long int y;

unsigned long int tipining qiymatlari oralig'i 0 dan 4294967295 gacha. O'zgaruvchini e'lon qilishga misol: unsigned long int x;

long kalit so'zi butun tipdan oldin ikki marta ko'rsatilishi mumkin. long long int va signed long long int tiplarining qiymatlari oralig'i -9223372036854775808 dan 9223372036854775807 gacha. O'zgaruv-chilarni e'lon qilishga misol: long long int x; signed long long int y;

unsigned long long int tipining mumkin bo'lgan qiymatlari oralig'i 0 dan 18446744073709551615 gacha. O'zgaruvchini e'lon qilishga misol: unsigned long long int x;

long kalit so'zini double tipidan oldin ko'rsatish ham mumkin. O'zgaruvchini e'lon qilishga misol: long double z;

Modifikatorlardan foydalanishda int tipi ko'rsatilmaganda (по умолчанию) qabul qilingan, shu sababli int tipini ko'rsatmaslik mumkin.

```
short x; // short int x; ga teng
long y; // long int y; ga teng
signed z; // signed int z; ga teng
unsigned k; // unsigned int k; ga teng
```

Bir ko'rsatmada bir vaqtda berilganlar tipi nomidan keyin bir necha o'zgaruvchini vergul bilan ajratgan holda e'lon qilish mumkin:

```
int x, y, z;
```

Agar o'zgaruvchi o'z qiymatini tashqaridan o'zgartirishi mumkin bo'lsa, modifikatordan oldin volatile kalit so'zi ko'rsatiladi. Bu kalit so'z dasturni optimallashtirishning oldini oladi, bunda o'zgaruvchining qiymati faqat dasturda o'zgartirilishi mumkinligi nazarda tutiladi.

O'zgaruvchi e'lon qilinganida u uchun, foydalanilayotgan tip va operatsion tizimning razryadiga bog'liq bo'lgan, ma'lum bir hajmda xotira ajratiladi. Masalan, int tipi 16 bitli tizimda 16 bit, 32 bitlida – 32 bit, 64 bitlida esa – 64 bit. Mashinaga bog'liq bo'lmagan kodni hosil qilish uchun tip o'lchamini sizeof operatori yordamida aniqlash lozim. Operator ikkita formatga ega:

```
<O'lcham> = sizeof<O'zgaruvchi>;
<O'lcham> = sizeof(<Berilganlar tipi>);
```

Berilganlar tipi qavs ichida ko'rsatilishi lozim, o'zgaruvchining nomini ham qavs ichida, ham qavssiz ko'rsatish mumkin. Misol:

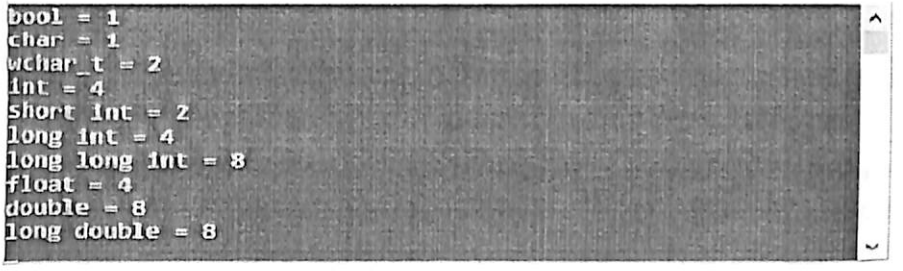
```
int x;
cout << sizeof x << endl;
cout << sizeof(x) << endl;
cout << sizeof(bool) << endl;
```

sizeof operatoridan foydalanishga misol sifatida berilganlarning barcha tiplari o'lchamlarini chop qiluvchi dastur listinggini keltiramiz.

```
#include <iostream>
using namespace std;
int main() {
```

```
cout << "bool = " << sizeof(bool) << endl;
cout << "char = " << sizeof(char) << endl;
cout << "wchar_t = " << sizeof(wchar_t) << endl;
cout << "int = " << sizeof(int) << endl;
cout << "short int = " << sizeof(short int) << endl;
cout << "long int = " << sizeof(long int) << endl;
cout << "long long int = "<< sizeof(long long int) << endl;
cout << "float = " << sizeof(float) << endl;
cout << "double = " << sizeof(double) << endl;
cout << "long double = " << sizeof(long double) << endl;
cin.get();
return 0;
}
```

32-bitli Windows XP operatsion tizimida dasturning bajarilishida quyidagicha natija chiqariladi (o'lcham baytlarda ko'rsatilgan):



1.4. Konsol oynasi berkilishining oldini olish

Yaratilgan dastur [Ctrl]+[F5] klavishlar kombinatsiyasi orqali ishga tushirilganda satr oxiriga avtomatik ravishda «Для продолжения нажмите любую клавишу...» satri qo'yiladi. har qanday klavishni bosish konsol oynasining yopilishiga olib keladi. Ammo foydalanuvchi dasturni bunday ishga tushurmaydi. Ishga tushirish buyruqlar satridan amalga oshirilsa, foydalanuvchi natijani ko'rishi mumkin, Fayl nishonida sichqoncha tugmasi ikki marta bosilib ishga tushirilsa, konsol oynasi

ochiladi va tezda yopiladi. Oyna yopilmasligi uchun klavishni bosishni kutib turuvchi ko'rsatma qo'yish zarur. Buni bir necha usulda bajarish mumkin.

Birinchi usul – _getch() funksiyasidan foydalanish. Funksiyadan foydalanishdan avval conio.h faylini ulash zarur. Funksiya qaytaradigan qiymatni e'tiborga olmaslik mumkin.

Funksiya prototipi:

```
#include <conio.h>
int _getch(void);
```

_getch()funksiyasidan foydalanishga misol:

```
#include <iostream>
#include <conio.h> // _getch() uchun
int main() {
// Ko'rsatmalar
std::cout << "<Enter> klavishini bosing...";
_getch();
return 0;
}
```

Ikkinchi usul – system() funksiyasidan foydalanishga asoslangan. Bu funksiya buyruqni operatsion tizimga uzatish imkonini beradi. "Для продолжения нажмите любую клавишу..." satrini chiqarish va klavishni bosishni kutib turish uchun pause buyrug'i tayinlangan. Funksiyadan foydalanishdan avval cstdlib (yoki stdlib.h) faylini ulash zarur bo'ladi.

```
#include <cstdlib>
int system(const char *Command);
```

system() funksiyasidan foydalanish:

```
#include <iostream>
#include <cstdlib> // system() uchun
int main() {
// Ko'rsatmalar
system("pause") ;
return 0; }
```

Yuqorida keltirilgan ikki usul uchun qo'shimcha fayl ulashga to'g'ri keladi. Bundan tashqari, _getch() funksiyasi kompilyator tomonidan ta'minlanmasligi, uning o'rniga getch() funksiyasidan foydalanishga to'g'ri kelishi mumkin. system() funksiyasi ko'plab ortiqcha amallarni bajaradi. Bu usullar o'rniga cin obyektining get() metodidan foydalanish yaxshi usul. Bu metod kiritilgan belgini olish imkonini beradi. Kiritish <Enter> klavishi bosilgandan keyin amalga oshiriladi. Funksiya qaytaradigan qiymatni e'tiborga olmaslik mumkin.

Funksiya prototipi quyidagicha ko'rinishga ega:

```
#include <iostream>
int get();
```

get() metodidan foydalanilganida bir narsani hisobga olish lozim: agar dasturda berilganlarni kiritish amalga oshirilgan bo'lsa, buferda belgilar qolib ketishi mumkin. Bu holda birinchi belgi avtomatik ravishda get() metodiga uzatiladi va konsol oynasi darhol berkiladi. Shu sababli, berilganlarni kiritgandan so'ng, qo'shimcha ravishda ignore() metodidan foydalanish zarur. Metodning prototipi quyidagicha:

```
#include <iostream>
istream &ignore(streamsize Count=1, int_type
Metadelim=EOF);
```

Birinchi parametrda buferdan o'qiluvchi belgilarning maksimal soni beriladi (ko'rsatilmaganda bir belgi). Ikkinchi parametr belgi-ajratkichni beradi. Agar belgi-ajratgich ko'rsatilgan sondagi belgilar o'qilmasdan oldin uchrasa, o'qish to'xtatiladi. Metod chiqarish oqimiga murojaat qaytaradi. Ko'pgina hollarda, metod nomini parametrsiz ko'rsatib, bir belgini qoldirish (odatda satrni ko'chirish belgisi) yetarli: cin.ignore().get();

Ammo, buferda boshqa belgilar qolishi ham mumkin. Shu sababli yaxshisi birinchi parametrda bir necha belgini (masalan, '200') e'tiborga olmaslikni, ikkinchi parametrda satrni ko'chirish belgisini ko'rsatish lozim:

```
cin.ignore(200,'\n').get();
```

Agar berilganlar kiritilmagan bo'lsa, u holda faqat get() metodidan foydalanish yetarli. get() va ignore() metodlarini qo'llashga misol.

```
#include <iostream>
int main() {
 // Ko'rsatmalar
 cout << "<Enter> klavishini bosing... ";
 cin.get(); // Agar berilganlar kiritilmagan bo'lsa
  // Agar berilganlar kiritilgan bo'lsa u holda: cin.
ignore(200, '\n').get();
 return 0;
}
```

1.5. Dastur kodida izohlar

Izohlar dastur matnida tushuntirishlar qo'yish uchun mo'ljallangan va kompilyator ularni to'la e'tiborga olmaydi. Izoh ichida ihtiyoriy matn joylashishi mumkin (bajarilishi talab qilinmaydigan ko'rsatmalar ham). Izohlar kompilyator uchun emas, balki, dasturchi uchun kerak. Kodda izohlardan foydalanish kod qismining tayinlanishini tezda eslash imkonini beradi.

Ikki xil izoh mavjud: bir satrli va ko'p satrli. Bir satrli izohlar "//" belgilaridan boshlanadi va satr oxirida tugaydi. Bunday izohlarni satr boshida yoki ko'rsatmadan keyin berish mumkin. Agar ko'rsatmadan oldin qo'yilsa, u bajarilmaydi. Agar "//" belgilari qo'shtirnoq ichida berilgan bo'lsa, ular izohning boshlanishi hisoblanmaydi. Bir satrli izohga misollar:

```
cout << "Salom, Olam!"; // bu satr izoh emas
//cout << "Salom, Olam!"; // ko'rsatma bajarilmaydi
char s[] = "//Bu izoh emas!!!";
```

Ko'p satrli izohlar "/*" belgisidan boshlanadi va "*/" belgisi bilan tugaydi. Bunday izoh bir yoki bir necha satrda joylashishi mumkin. Ko'p satrli izohni ifodaning ichiga joylashtirish ham mumkin. Ko'p satrli izohlar ichma-ich joylashishi mumkin

emas, bir satrli izohlar esa ko'p satrli izoh ichida bo'lishi mumkin. Ko'p satrli izohga misollar:

```
/*...
Ko'p satrli izoh
...*/
cout << "Salom, Olam!"; /*Bu izoh*/
/*cout<< "Salom, Olam!"; ko'rsatma bajarilmaydi*/
int x;
x = 10 /*Izoh*/ + 50 /* ifoda ichida*/ ;
char s[] = "/*Bu izoh emas!!!*/";
```

VC++ redaktori kod fragmentini tezda izohga almashtirish imkonini beradi. Buning uchun bir (yoki bir nechta) ko'rsatma ajratiladi va Правка menyusida *"Дополнительно*|Закомментировать выделенные строки" punkti tanlanadi. Bundan tashqari, "qaynoq tugma"lardan ham foydalanish mumkin. Buning uchun avval [Ctrl]+[K] klavishlar kombinatsiyasi, so'ngra [Ctrl]+[C] klavishlar kombinatsiyasi bosiladi. Agar satr qismi ajratilgan bo'lsa, u ko'p satrli izohga almashtiriladi.

Izohni olib tashlash uchun Правка menyusida "Дополнительно| Раскомментировать выделенные строки" punkti tanlanadi. "Qaynoq tugma"lardan foydalanilsa, avval [Ctrl]+[K] klavishlar kombinatsiyasi, so'ngra [Ctrl]+[U] klavishlar kombinatsiyasi bosiladi.

2. O'zgaruvchilar va ifodalar

2.1. O'zgarmaslar

O'zgarmaslar – dasturning bajarilishi jarayonida qiymati o'zgarmaydigan o'zgaruvchilardir. O'zgarmas deyilganda, keng ma'noda o'zgartirish mumkin bo'lmagan har qanday qiymat tushuniladi, masalan, 10, 12.5, 'W', "string".

O'zgarmasni e'lon qilishda berilganlar tipidan oldin const kalit so'zi ko'rsatiladi: const int X = 5;

O'zgarmasning nomida faqat katta harflardan foydalanish qabul qilingan. Bu dastur ichida o'zgarmasni oddiy o'zgaruvchidan farqlash imkonini beradi. O'zgarmas e'lon qilinganidan so'ng, undan ifodalarda foydalanish mumkin.

```
const int X = 5; int y; y = X + 20;
```

O'zgarmasni #define direktivasi yordamida ham yaratish mumkin. Bu direktivada ko'rsatilgan qiymat, kompilyatsiyaga qadar ifodaga ko'chiriladi. #define direktivasida ko'rsatilgan nomni **makroaniqlanish** yoki **makros** deb nomlash qabul qilingan. Direktiva quyidagicha formatga ega:

#define <Makros nomi> <Qiymat>

#define ko'rsatmasidan foydalanishga misol:

```
#include <iostream>
#define X 5
int main() {
 int y;
 y = X + 20;
 cout << y << endl;
 cin.get();
 return 0;
}
```

#define direktivasida "=" operatoridan foydalanilmaydi va ko'rsatma oxirida nuqtali vergul ko'rsatilmaydi. Agar nuqtali vergul ko'rsatilsa, u ifodaga qo'yiladi. Masalan, makros quyidagicha aniqlansa: #define X 5;

u holda qiymat qo'yilganidan so'ng, y = X + 20;

ifoda quyidagicha ko'rinishga ega bo'ladi: y = 5; +20;

'5' raqamidan keyin turgan nuqtali vergul ko'rsatmaning oxiri hisoblanadi. Shu sababli, "y" o'zgaruvchiga '25' emas, '5' qiymat oladi. Bunday holatlar kompilyatsiya jarayonida xatolikni keltirib chiqarmaganligi sababli (+20; ko'rsatmasi) topish qiyin bo'lgan xatoliklarga olib keladi.

Makrosning qiymati sifatida butun ifodani ko'rsatish mumkin:

#define X 5 + 5

Bunday e'lon qilish ham noto'g'ri. Bunda hech qanday hisoblash amalga oshirilmaydi. Agar ifoda y = X * 20; ko'rinishga ega bo'lsa, u holda qiymat qo'yilganidan so'ng, ifoda quyidagicha ko'rinish oladi: y = 5 + 5 * 20;

Shunday qilib, natija '200' soni o'rniga '105'ga teng bo'ladi.

Makrosning qiymati sifatida qo'shtirnoq ichida satrni ko'rsatish mumkin:

#define ERR "Xatolik haqida xabar"

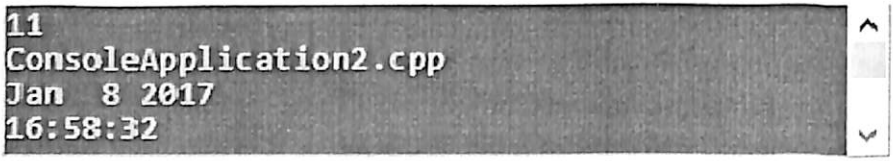
Uzun satr ko'rsatilganda makrosning aniqlanishi bitta satrda ekanligini unutmaslik lozim. Agar qiymatni bir necha satrga joylashtirish zarurati paydo bo'lsa, satr oxirida teskari og'ma chiziq qo'shish lozim. Og'ma chiziqdan keyin hech qanday belgi bo'lmasligi lozim (izohlar ham). Makrosni bir necha satrga joylashtirishga misol:

```
#define ERR "Xatolik haqida xabar\
bir necha \
satrda"
```

C++ tilida biriktirilgan makroslar mavjud (nomdan oldin va keyin ikkita ostki chiziq):

__FILE__ – fayl nomi;
__LINE__ – joriy satr nomeri (tartibi);
__DATE__ – faylning kompilyatsiya kuni;
__TIME__ – faylning kompilyatsiya vaqti.

```
#include <iostream>
int main() {
cout << __LINE__ << endl;
cout << __FILE__ << endl;
cout << __DATE__ << endl;
cout << __TIME__ << endl;
cin.get();
return 0;
}
```



Bu makroslardan tashqari turli sarlavha fayllarida e'lon qilingan ko'plab boshqa makroslar mavjud.

2.2. O'zgaruvchilar

O'zgaruvchilar – berilganlarni saqlash uchun foydalaniluvchi xotira qismi. O'zgaruvchidan foydalanishdan avval uni *global* (funksiyalardan tashqarida) yoki *lokal* (funksiya ichida) e'lon qilish lozim. Ko'pgina hollarda funksiyaning e'loni uning aniqlanishi ham bo'ladi.

Glabal o'zgaruvchilar fayldagi barcha funksiyalaring ichida, lokal o'zgaruvchilar esa faqat e'lon qilingan funksiyaning ichidagina ko'rinadi. O'zgaruvchini e'lon qilish uchun quyidagicha formatdan foydalaniladi:

[<Spesifikator>][<Modifikator>]<Tip><O'zgaruvchi1>
[=<Qiymat1>][, ..., <O'zgaruvchiN>[=<QiymatN>]];

Har bir o'zgaruvchi lotin harflari, raqamlar, ostki chiziqlardan tashkil topgan yagona nomga ega bo'lishi lozim. Bunda o'zgaruvchining nomi raqamdan boshlanmasligi kerak. O'zgaruvchining nomini ko'rsatishda registrni hisobga olish muhim: "x" va "X" – turli o'zgaruvchilar.

Nom sifatida kalit so'zlardan foydalanish mumkin emas. Kalit so'zlar ro'yxati quyida jadvalda keltirilgan. Ba'zi kompilyatorlarda qo'shimcha kalit so'zlar e'lon qilingan bo'lishi mumkin. Barcha kalit so'zlarni yoddan bilish shart emas, O'zgaruvchining nomi sifatida kalit so'zlardan foydalanishga harakat kompilyatsiya jarayonida xatolikka olib keladi.

O'zgaruvchilarning to'g'ri nomlari: x, y1, str_name, strName.

O'zgaruvchilarning noto'g'ri nomlari: 1x, Noma'lum, Переменная.

C++ tilining kalit so'zlari

asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	tty
case	extern	public	typed ef
catch	false	register	typeid
char	float	reinterpret_ cast	typename
class	for	return	union
const	friend	short	unsigned
const_cast	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void
delete	int	static_cast	volatile
do	long	struct	wchar_t
double	mutable	switch	while
dynamic_cast	namespace	template	

O'zgaruvchi e'lon qilganida, «=» o'zlashtirish operatoridan keyin bo'shlang'ich qiymatini berish ham mumkin. Bu amal o'zgaruvchini nomlash deyiladi. Nomlashda qiymatni qavs ichida ko'rsatish ham mumkin. O'zgaruvchining qiymatini ko'rsatishga misol:

```
int x, y = 10, z = 30, к;
int x(10); // «x» o'zgaruvchi 10 soniga teng
```

O'zgaruvchi e'lon qilinganidan so'ng ko'rinarli bo'ladi, shu sababli bu o'zgaruvchidan bir satrda (verguldan keyin) boshqa o'zgaruvchilarni nomlash uchun undan foydalanish mumkin.

```
int x = 5, y = 10, z = x + y; // «z» 15 soniga teng
```

Global (funksiyadan tashqarida e'lon qilingan) o'zgaruvchilarni nomlash faqat bir marta amalga oshiriladi. Lo-

kal (funksiya ichida e'lon qilingan) o'zgaruvchilar har bir chaqirishda nomlanadi, statik lokal o'zgaruvchilar esa (o'z qiymatini chaqirishlar oralig'ida saqlab turadi) – bir marta funksiyaning birinchi chaqirilishida.

E'lon qilishda o'zgaruvchiga qiymat berilmagan bo'lsa, u holda:

– global o'zgaruvchilar avtomatik ravishda '0' qiymat oladi;

– lokal o'zgaruvchilarga qiymat o'zlashtirilmaydi. O'zgaruvchi "musor" deb nomlanuvchi ihtiyoriy qiymat oladi;

– statik lokal o'zgaruvchilar avtomatik ravishda '0' qiymat oladi.

bool tipiga ega bo'lgan o'zgaruvchiga true yoki false qiymatini berish mumkin. true qiymati '1' soniga mos keladi, false – qiymat '0' soniga, true va false o'rniga son qiymatlarini ko'rsatish ham mumkin:

```
bool a, b, c, d;
a = true; b = false;
c = 1; // ekvivalenti: c=true;
d = 0; // ekvivalenti: d=false;
```

char tipiga ega o'zgaruvchiga sonli qiymat (belgi kodi) berish yoki belgini apostrof ichida ko'rsatish mumkin. Qo'shtirnoq ishlatish mumkin emas, bu holda bir belgi o'rniga ikki belgi bo'ladi: berilgan belgi va nol belgi.

```
char ch1, ch2;
ch1 = 119; // "w" harfi
ch2 = 'w'; // "w" harfi
```

Apostroflar ichida maxsus belgilarni ko'rsatish mumkin. Maxsus belgilar – bu xizmatchi va chop qilinmaydigan belgilarni ifodalovchi belgilar kombinatsiyasi. C++ tilida mumkin bo'lgan maxsus belgilarni keltirib o'tamiz:

\0 – nol belgi;	\' – apostrof;
\n – yangi sartrga o'tish;	\f – formatni ko'chirish;
\r – karetkani qaytarish;	\» – qo'shtirnoq;
\t – gorizontal tabulyatsiya;	\? – so'roq belgisi;
\v – vertikal tabulyatsiya;	\\ – teskari og'ma chiziq;
\a – tovush signali;	\N – N ning sakkizlik qiymati;
\b – bir belgiga qaytish;	\xN – N ning o'n oltilik qiymati.

Sakkizlik va o'n oltilik qiymatlarni ko'rsatishga misol:

```
char ch1, ch2;
ch1 = '\167'; // "w" (sakkizlik qiymati) harfi
ch2 = '\x77'; // "w" (o'n oltilik qiymati) harfi
```

wchar_t tipiga ega o'zgaruvchiga qiymat char tipidagi kabi o'zlashtiriladi, faqat ochiluvchi apostrofdan oldin "L" harfi ko'rsatiladi:

```
wchar_t ch; ch = L'w';
```

Butun qiymat o'nlik, sakkizlik yoki o'n oltilik shaklda beriladi. Sakkizlik sonlar noldan boshlanadi va '0'dan '7'gacha raqamlarni oladi. O'n oltilik sonlar '0x' (yoki '0X') belgilar kombinatsiyasidan boshlanadi va '0'dan '9'gacha raqam va 'A'dan 'F'gacha harflarni (registrning ahamiyati yo'q) oladi. Sakkizlik va o'n oltilik qiymatlar o'nlik qiymatga almashtiriladi. Misol:

```
int x, y, z;
x = 119; // O'nlik qiymat
y = 0167; // Sakkizlik qiymat
z = 0x77; // O'n oltilik qiymat
```

Ko'rsatilmaganda butun o'zgarmaslar signed int tipiga ega. Agar tipni o'zgartirish zarur bo'lsa, sondan keyin quyidagi harflardan biri ko'rsatiladi:

– L (yoki l) – long int tipi. Qiymatni ko'rsatishga misol: 10L;

– U (yoki u) – unsigned int tipi. Qiymatni ko'rsatishga misol: 10U;

– agar U va L harflari bir vaqtda ko'rsatilsa, tip unsigned long int bo'ladi. Qiymatni ko'rsatishga misol: 10UL.

Haqiqiy son nuqta va (yoki) 'E' harfidan boshlanuvchi eksponentani olishi mumkin (registr ahamiyatga ega emas):

float x, y; double z, к;

x = 20.0; y = 12.1e5; z = .123; к = 4.7E-5;

O'zgaruvchilarga tip nomidan keyin qavslarni ko'rsatib, nol qiymat o'zlashtirish mumkin:

int x = int(); float x = float(); double y = double();

Haqiqiy o'zgarmaslar ko'rsatilmaganda double tipiga ega. Uni boshqa tipga almashtirish zarur bo'lsa, sondan keyin quyidagi harflar ko'rsatiladi:

- F (yoki f) – float tipi. Qiymatni ko'rsatishga misol: 12.3f;
- U (yoki u) – long double tipi. Qiymat nuqta va (yoki) eksponentani olishi lozim, aks holda long int tipi bo'ladi. Misol: 12.3L.

Vergul operatori bir ko'rsatma ichida bir vaqtda bir necha ifodalarni joylashtirish imkonini beradi. Masalan: int x, y;

Oxirgi hisoblashlar natijasini qandaydir boshqa o'zgaruvchiga o'zlashtirish mumkin. Bunda ifodalar qavs ichida bo'lishi lozim, chunki, vergul operatorining ustunlik darajasi o'zlashtirish operatori ustunlik darajasidan past. Misol:

```
int x = 0, y = 0, z = 0;
x = (y = 10, z = 20, y + z);
cout << x << endl; // 30
cout << y << endl; // 10
cout << z << endl; // 20
```

Bu misolda o'zgaruvchilar nomlanganidan va e'lon qilinganidan so'ng "y"ga '10' qiymati, so'ngra "z"ga '20' qiymati o'zlashtiriladi. Keyin "y+z" ifoda hisoblanadi oshiriladi va natijasi "x" o'zgaruvchiga o'zlashtiriladi.

2.3. O'zgaruvchilarning ko'rinish sohasi

O'zgaruvchidan foydalanishdan avval, u oldindan e'lon qilinishi lozim. Ularni **global** (funksiyadan tashqarida) yoki **lokal** (funksiya yoki blok ichida) e'lon qilish mumkin.

Global o'zgaruvchilar – bu dasturda funksiyadan tashqarida e'lon qilingan o'zgaruvchilar. Global o'zgaruvchilar butun dasturda ko'rinadi (funksiyalarda ham). Bunday o'zgaruvchilar faqat bir marta nomlanadi. E'lon qilishda global o'zgaruvchiga qiymat o'zlashtirilmasa, u holda avtomatik ravishda nol (0) qiymat bilan nomlanadi.

Lokal o'zgaruvchilar – bu funksiya yoki blok ichida (figurali qavslar ichida) e'lon qilingan o'zgaruvchilar. Lokal o'zgaruvchlar funksiya yoki blok ichida ko'rinadi. Bunday o'zgaruvchilarni nomlash funksiyaning har bir chaqirilishida blokka kirishda amalga oshiriladi. Funksiya yoki blokdan chiqqandan so'ng lokal o'zgaruvchi o'chiriladi. E'lon qilishda lokal o'zgaruvchiga qiymat berilmagan bo'lsa, u holda o'zgaruvchi "**musor**" (yaroqsiz ma'lumot) deb nomlanuvchi ihtiyoriy qiymat oladi. Statistik lokal o'zgaruvchilar bundan mustasno, chunki ularga avtomatik ravishda *nol* qiymat o'zlashtiriladi va funksiyadan chiqishda qiymatni saqlab qoladi.

Agar lokal o'zgaruvchining nomi global o'zgaruvchi nomi bilan mos kelsa, u holda funksiya ichidagi barcha amallar lokal o'zgaruvchi bilan amalga oshiriladi, global o'zgaruvchining qiymati esa o'zgarmaydi. Bunday hollarda global o'zgaruvchiga kirish uchun o'zgaruvchi nomidan oldin ikkita ikki nuqta ko'rsatish zarur, masalan, ::x.

Global va lokal o'zgaruvchilarning ko'rinish sohasi misol tariqasida quyidagi listingda keltirilgan:

```
// O'zgaruvchilarning ko'rinish sohasi
#include <iostream>
int x = 10; // Global o'zgaruvchi
void func();
int main() {
 func();
 // Bu yerda func() funksiyasida x va y o'zgaruvchilar
ko'rinmaydi
```

```
// x global o'zgaruvchining qiymatini chiqarish
cout << x << endl; // 10
{ // Blok
int z = 30;
cout << z << endl; // 30
}
for (int i = 0; i < 10; ++i) { // Bu yerda "z" o'zgaruvchi
ko'rinmaydi
cout << i << endl;
}
cin.get(); // Bu yerda "i" o'zgaruvchi ko'rinadi
return 0;
}
void func() {
// Bu yerda "z" o'zgaruvchi ko'rinmaydi
int x = 5, y = 20; // Lokal o'zgaruvchilar
// "x" lokal o'zgaruvchining qiymatini chiqarish
cout << x << endl; // 5
// "x" global o'zgaruvchining qiymatini chiqarish
cout <<::x << endl; // 10
}
```

Qandaydir shartga bog'liq holda o'zgaruvchiga qiymat berilgan bo'lsin:

```
// Noto'g'ri !
if(x == 10) { // Qandaydir shart
int y; y = 5;
} else {
int y;
y = 25;
} // Bu yerda "y" o'zgaruvchi aniqlanmagan!
```

Bu misolda "y" o'zgaruvchi faqat blokning ichida ko'rinadi. Shart operatori if dan keyin o'zgaruvchi mavjud emas. O'zgaruvchi blok ichida va undan chiqqandan keyin ham ko'rinishi uchun blokdan avval e'lon qilinishi lozim:

```
// To'g'ri
int y;
if(x == 10) { // Qandaydir shart
 y = 5;
 else {
 y = 25;
 }
```

Bir nomli lokal o'zgaruvchini e'lon qilishdan avval funksiya yoki blok ichida murojaat qilinayotgan bo'lsa, u holda murojaat global o'zgaruvchidan foydalanadi, e'londan keyin – lokal o'zgaruvchidan. Agar bu nomdagi global o'zgaruvchi mavjud bo'lmasa, kompilyatsiya jarayonida xatolik bo'ladi. Masalan:

```
#include <iostream>
int x = 10; // Global o'zgaruvchi
int main() {
 // Global o'zgaruvchi qiymati chiqariladi
 cout << "x = " << x << endl; // 10
 int x = 5; // Lokal o'zgaruvchi
 // Lokal o'zgaruvchi qiymati chiqariladi
 cout << "x = " << x << endl; // 5
 cin.get();
 return 0;
 }
```

```
x = 10
x = 5
```

2.4. Berilganlar tipini aniqlash

Dasturning bajarilishida berilganlar tipini aniqlash uchun typeid operatoridan foydalaniladi. Operatordan foydalanish uchun typeinfo faylini ulash zarur:

```
#include <typeinfo>
```

Operator quyidagicha formatga ega:

<type_info sinfi namunasi> = typeid(<Berilganlar>);

typeid operatori qiymat sifatida type_info sinfi namunasini qaytaradi. Bu namunani "==" (teng) va "!=" (teng emas) operatorlaridan foydalangan holda shu sinfning boshqa namunalari bilan taqqoslash mumkin. name() metodi berilganlar tiplari nomlanishini olish imkonini beradi. Listingda typeid operatoridan foydalanishga misol keltirilgan.

```
// Berilganlar tipini dinamik aniqlash
#include <iostream>
#include <typeinfo>
int main() {
long int x = 5;
int y = 12;
double z = 3.14;
cout << typeid(x).name() << endl; // long
cout << typeid(y).name() << endl; // int
cout << typeid(z).name() << endl; // double
cout << typeid(12.1f).name() << endl; // float
if(typeid(x) == typeid(y)) {
cout << "==" << endl;
} else {
cout << "!= " << endl; // !=
}
if(typeid(x) != typeid(y)) {
cout << "!= " << endl; // !=
} else {
cout << "==" << endl;
}
cin.get();
return 0;
}
```

2.5. typedef operatori. Tiplarni keltrish

typedef operatori mavjud berilganlar tipi uchun *yangi tip* yaratish imkonini beradi. Keyinchalik o'zgaruvchini e'lon qilishda yangi tipni ko'rsatish mumkin. Operatorning formati quyidagicha:

typedef <mavjud tip> <yangi tip>;

Masalan, long int tipi uchun yangi tip yaratamiz:

typedef long int lint;

lint x = 5, y = 10;

Yangi tip yaratilgandan so'ng, uning nomidan boshqa yangi nom yaratishda foydalanish mumkin:

typedef long int lint;

typedef lint newint;

newint x = 5, y = 10;

Yangi tiplar mashinaga bog'liq bo'lmagan dasturlarni yaratish uchun mo'ljallangan. Dasturni boshqa kompyuterga ko'chirishda bir satrni o'zgartirish yetarli bo'ladi. Bunday yondoshuvdan ko'pincha standart kutubxonada foydalaniladi. Masalan, satr uzunligini olish imkonini beruvchi strlen() funksiyasi prototipi quyidagicha ko'rinishga ega:

size_t strlen(const char *Str);

Bu prototipda strlen() funksiyasi qaytaruvchi size_t berilganlar tipi yangi tip emas, unsigned int tipi uchun **yangi tip** hisoblanadi.

Tiplarni keltirish. O'zgaruvchi e'lon qilinganida berilganlarning qandaydir tipini ko'rsatish zarur. Keyinchalik o'zgaruvchi ustida bunday tip uchun tayinlangan amallarni bajarish mumkin. Ifodada turli tipdagi o'zgaruvchilardan foydalanil-

sa, ifoda natijasining tipi eng murakab tipga mos keladi. Masalan, int va double tipidagi o'zgaruvchilarni qo'shish amalga oshirilayotgan bo'lsa, u holda natija avtomatik ravishda haqiqiy tipga almashtiriladi. Bu ifoda natijasi double tipidagi qiymat bo'ladi.

O'zgaruvchiga mumkin bo'lmagan qiymatni o'zlashtirishga harakat qilishda kompilyator xatolik haqida (agar avtomatik almashtirish mumkin bo'lmasa) yoki ogohlantiruvchi xabar beradi (qiymat qirqilganida). Masalan, satr o'zgaruvchiga butun tip berishga harakat qilinganda tuzatib bo'lmaydigan xatolikka olib keladi (int x = "string";).

«error C2440: инициализация: невозможно преобразовать «const char[7]» в «int»».

Haqiqiy son butun o'zgaruvchiga o'zlashtirilganda quyidagicha ogohlantiruvchi xabar chiqariladi:

«warning C4244: инициализация: преобразование «double» в «int», возможна потеря данных».

Masalan, int x = 10.5; – bunda sonnig kasr qismi tashlab yuboriladi va o'zgaruvchiga '10' soni o'zlashtiriladi va dasturning bajarilishi davom ettiriladi. Bunday xabar nuqtadan keyin nol turganda ham chiqariladi. Bunday xabarlardan holi bo'lish uchun tiplarni aniq keltirishni bajarish zarur. VC++ tilida tiplarni keltirish uchun quyidagi operatorlar qo'llaniladi:

a) static_cast – tiplarni standart keltirish uchun foydalaniladi. Operatorning formati: static_cast<Natija tipi>(Ifoda)

```
int x = 10, y = 3;
cout << x / y << endl; // 3
cout << static_cast<double>(x) / y << endl; // 3.33333
```

b) reinterpret_cast – ko'rsatkichni umuman boshqa tipga keltirish uchun foydalaniladi, hattoki, mumkin bo'lmaganiga ham.

Formati: reinterpret_cast<Natija tipi>(Ifoda)

c) const_cast – const va volatile kalit so'zlarining harakatlarini bekor qiladi. Formati: const_cast<Natija tipi>(Ifoda)

Funksiya ichida o'zgarmas ko'rsatkichni oddiyga keltirishga misol:

```
void func(const int *x) {
int *p = const_cast<int *>(x);
// x = 50; // Xato!!!
*p = 30; // Endi qiymatni o'zgartirish mumkin
}
```

Funksiya ichida murojaatdan const kalit so'zining ta'sirini o'chirish:

```
void func(const int &x) {
// x = 50; // Xato!!!
 // Endi qiymatni o'zgartirish mumkin
const_cast<int &>(x) = 30;
}
```

d) dynamic_cast – ko'rsatkichlar va murojaatlar tiplarini keltirishni bajaradi. Polimorf tiplarni keltirish uchun qo'llaniladi. Ko'rsatkichni keltirish samarasiz yakunlansa, operator nolinchi ko'rsatkich qaytaradi. Murojaat bilan muammo yuzaga kelganda esa, bad_cast istesnosi generatsiya qilinadi (istesno obyekti typeinfo funksiyasida e'lon qilingan). Formati: dynamic_cast<Natija tipi>(Ifoda)

dynamic_cast operatori obyektga yo'naltirilgan dasturlashda yanada kengroq tushuntiriladi.

C++ tili C tilida foydalaniluvchi tiplarni keltirishni ham ta'minlaydi. Keltirish formati: (Natijalar tipi)Ifoda

Butun sonni double tipidagi haqiqiy songa almashtirishga misol:

```
int x = 10, y = 3;
cout << x / y << endl; // 3
cout << (double)x / y << endl; // 3.33333
```

Ifodani tip nomidan keyin qavs ichida berish ham mumkin:

```
cout << double(10 / 3) << endl; // 3.33333
```

3. Sonlar bilan ishlash

3.1. Sonlarning tiplari

C++ tilida berilganlarning barcha elementar tiplari (bool, char, wchar_t, int, float va double) amaliy jihatdan sonli hisoblanadi.

bool tipi faqat true va false o'zgarmaslariga mos keluvchi '1' va '0' qiymatlar olishi mumkin. char va wchar_t tiplari belgining o'zini emas, balki uning kodini oladi. Shu sababli, bu tiplardan int, float va double tiplarini olgan ifodalarda birgalikda foydalanish mumkin. Masalan:

```
bool a = true; // 1
char ch = 'w'; // 119
cout << (a + ch + 10) << endl; // 130
```

Butun sonlarni saqlash uchun int tipi tayinlangan. Tipning qiymatlari oralig'i (diapozoni) operatsion tizimning razryadiga bog'liq. 16 bitli operatsion tizimda bu diapozon: -32768 dan 32767 gacha. 32 bitli operatsion tizimda: -2147483648 dan 2147483647 gacha.

short, long va long long kalit so'zlari yordamida int tipining aniq o'lchamimi ko'rsatish mumkin. Bu kalit so'zlardan foydalanilganda int tipi tushuniladi, shu sababli uni ko'rsatish shart emas.

short tipi 2 bayt joy band qiladi (-32768 dan 2767 gacha oraliq);

long tipi – 4 bayt (-2147483648 dan 2147483647 gacha oraliq);

long long tipi esa 8 bayt band qiladi (qiymatlari oralig'i: -9223372036854775808 dan 9223372036854775807 gacha).

VC++ tilida short, long va long long tiplari o'rniga mos ravishda __int16, __int32 va __int64 tiplaridan foydalanish mumkin (tipdan oldin ikkita ostki chiziq).

Ko'rsatilmaganda (по умолчанию) butun tiplar ishorali hisoblanadi. Sonnig ishorasi yuqori razryadda saqlanadi: 0 musbat

son, 1 esa manfiy son ekanligini bildiradi. unsigned kalit so'zi yordamida son nomanfiy ekanligini ko'rsatish mumkin. unsigned short tipi 0 dan 65535 gacha diapozonda sonlarni olishi mumkin.

unsigned long tipi – 0 dan 4294967295 gacha, unsigned long long tipi – 0 dan 18446744073709551615 gacha. Qiymatni signed tipidan unsigned tipiga almashtirishda ishora biti (agar son manfiy bo'lsa, 1 qiymat oladi) katta sonlarni keltirib chiqarishi mumkinligini hisobga olish lozim, chunki unsigned tipida katta bit ishora belgisini olgan:

```
int x = -1;
cout << (unsigned int)x << endl; // 4294967295
```

Butun sonlar o'nlik, sakkizlik yoki o'n oltilik shaklda berilishi mumkin. Sakkizlik sonlar noldan boshlanadi va 0 dan 7 gacha raqamlarni oladi. O'n oltilik sonlar 0x (yoki 0X) belgilar kombinatsiyasidan boshlanadi va 0 dan 9 gacha raqam va A dan F gacha (harflar registri ahamiyatga ega emas) harflarni olishi mumkin. Sakkizlik va o'n oltilik qiymatlar o'nli qiymatga almashtiriladi.

```
int x, y, z;
x = 119; // O'nlik qiymat
y = 0167; // Sakkizlik qiymat
z = 0x77; // O'n oltilik qiymat
```

Ko'rsatilmaganda butun o'zgarmaslar signed int tipiga ega. Agar tipni boshqa tipga o'zgartirish lozim bo'lsa, sondan keyin quyidagi harflar ko'rsatiladi: L (yoik l) – long int tipi. Qiymatni ko'rsatishga misol: 10L, U (yoik u) – unsigned int tipi. Qiymatni ko'rsatishga misol: 10U, agar u va l harflari bir vaqtda ko'rsatilgan bo'lsa u holda tip unsigned long int tipida bo'ladi. Qiymatni ko'rsatishga misol: 10ul.

Haqiqiy sonlarni saqlash uchun float va double tiplari tayinlangan. Haqiqiy son nuqta va (yoki) E (registr ahamiyatga ega emas) harfidan boshlanuvchi eksponentani olishi mumkin:

```
float x, y;
double z, k;
x = 20.0;
y = 12.1e5;
z = .123;
k = 47.E-5;
```

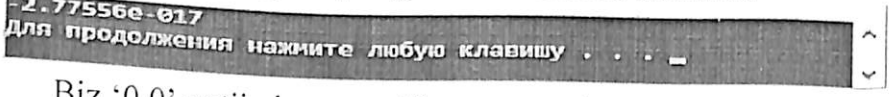
Ko'rsatilmaganda haqiqiy o'zgarmaslar double tipini oladi. Agar tipni boshqasiga o'zgartirish zarurati yuzaga kelsa, sondan keyin quyidagi harflardan biri ko'rsatiladi:

F (yoki f) –float tipi. Qiymatni ko'rsatishga misol: 12.3f;

L (yoki l) – long double tipi. Qiymat nuqta va (yoki) eksponentani olishi lozim, aks holda long int tipi bo'ladi. Misol: 12.3L.

Haqiqiy sonlar ustida amallar bajarishda hisoblashlar aniqligini hisobga olish zarur. Masalan, `cout << (0.3 – 0.1 – 0.1 – 0.1) << endl;`

ko'rsatmasining natijasi g'alati ko'rinishi mumkin:



Biz '0.0' natija kutgan edik, ammo misoldan ko'rinib turibdiki, umuman boshqa natija olindi (-2.77556e-017).

Agar ifodada turli tipdagi ma'lumotlardan foydalanilayotgan bo'lsa, u holda ifoda ancha murakkab bo'lgan tipga moslashtiriladi. Masalan, agar int tipidagi o'zgaruvchi double tipidagi o'zgaruvchi bilan qo'shilayotgan bo'lsa, butun son avtomatik ravishda haqiqiy tipga almashtiriladi. Bu ifodaning natijasi double tipiga ega bo'lgan qiymat bo'ladi. Ammo, ifodaning natijasi int tipidagi o'zgaruvchiga o'zlashtirilayotgan bo'lsa, double tipi int tipiga aylantiriladi (kompilyator ma'lumotlarning yo'qolishi mumkinligi haqida xabar chiqaradi):

```
int x = 10, y = 0;
double z = 12.5;
y = x + z;
cout << y << endl; // 22 (int tipi)
cout << (x + z) << endl; // 22.5 (double tipi)
```

3.2. Matematik konstantalar

VC++ tilining math.h sarlavha faylida matematik o'zgarmaslarning qiymatlarini olgan makroslar aniqlangan. Konstantalardan foydalanishda math.h faylini ulashdan oldin _USE_MATH_DEFINES deb nomlanuvchi makrosni aniqlab olish lozim.

Matematik o'zgarmaslarni keltirib o'tamiz:

M_PI – PI soni. Makrosning aniqlanishi:

#define M_PI *// 3.14159265358979323846*

M_PI_2 – pi/2 ifodaning qiymati. Makrosning aniqlanishi:

#define M_PI_2 *// 1.57079632679489661923*

M_PI_4 – pi/4 ifodaning qiymati. Makrosning aniqlanishi:

#define M_PI_4 *// 0.78539816339744830961 6*

M_1_PI – 1/pi ifodaning qiymati. Makrosning aniqlanishi:

#define M_1_PI *// 0.31830988618379067153 8*

M_2_PI – 2/pi ifodaning qiymati. Makrosning aniqlanishi:

#define M_2_PI *// 0.63 66197723675813430 76*

M_E – e konstantaning qiymati. Makrosning aniqlanishi:

#define M_E *// 2.71828182845904523536*

M_LOG2E – $\log_{2(e)}$ ifodaning qiymati. Makrosning aniqlanishi:

#define M_LOG2E *// 1.44269504088896340736*

M_LOG10E – $\log_{10}(e)$ ifodaning qiymati. Makrosning aniqlanishi:

#define M_LOG10E *// 0.43429448190325182765 1*

M_LN2 – ln(2) ifodaning qiymati. Makrosning aniqlanishi:

#define M_LN2 *// 0.69314718055994530941 7*

m_ln10 – Ln(10) ifodaning qiymati. Makrosning aniqlanishi:

#define M_LN10 *// 2.30258509299404568402*

m_2_sqrtpi – 2/sqrt(pi) ifodaning qiymati. Makrosning aniqlanishi:

#define M_2_SQRTPI *// 1.12837916709551257390*

M_SQRT2 – sqrt(2) ifodaning qiymati. Makrosning aniqlanishi:

#define M_SQRT2 *// 1.41421356237309504880*

M_SQRT1_2 – 1/sqrt(2) ifodaning qiymati. Makrosning aniqlanishi:

#define M_SQRT1_2 *// 0.70710678118654752440*

O‘zgarmaslarning qiymatlarini chiqarishga sodda misol ko‘raylik:

```
#include <iostream>
#define _USE_MATH_DEFINES
#include <math.h>
int main() {
 cout << M_PI << endl;
 cout << M_E << endl;
 cin.get();
 return 0;
 }
```

```
3.14159
2.71828
```

3.3. Sonlar bilan ishlash funksiyalari

Sonlar bilan ishlash uchun C++ tilida aniqlangan asosiy funksiyalar bilan tanishib chiqamiz:

abs() – absolyut qiymat qaytaradi. Funksiya prototiplari:

```
#include <cstdlib>
int abs(int X);
long abs(long X);
long long abs(long long X);
#include <cmath>
float abs(float X);
double abs(double X);
long double abs(long double X) ;
```

abs() funksiyasi long tipi uchun labs(), long long tipi uchun llabs(), float tipi uchun fabsf(), double tipi uchun fabs(), long double tipi uchun fabsl()funksiyasini chaqiradi. Misol:

cout << abs(-1) << endl; // *1*

pow() – sonni darajaga ko'tarish (X^Y). To'lish holida chiqishda mumkin bo'lgan qiymatdan tashqariga chiqish xatoligi yuzaga keladi. Funksiya prototiplari:

```
#include <cmath>
float pow(float X, int Y);
double pow(double X, int Y);
float pow(float X, float Y);
double pow(double X, double Y);
long double pow(long double X, int Y);
long double pow(long double X, long double Y);
```

Misol:

```
cout << pow(10.0,2) << endl; // 100
cout << pow(3.0,3.0) << endl; // 27
```

sqrt() – kvadrat ildiz. Funksiya prototiplari:

```
#include <cmath>
float sqrt(float X);
double sqrt(double X);
long double sqrt(long double X);
```

sqrt() funksiyasi float tipi uchun sqrtf(), long double tipi uchun sqrtl() funksiyasini chaqiradi. Funksiyadan foydalanishga misollar:

```
cout << sqrt(100.0) << endl; // 10
cout << sqrt(25.0) << endl; // 5
```

exp() – eksponenta. Funksiya prototiplari:

```
#include <cmath>
float exp(float X);
double exp(double X);
long double exp(long double X);
```

exp() funksiyasi float tipi uchun expf(), long double tipi uchun esa expl()funksiyasini chaqiradi.

log() – natural logarifm. Funksiya prototiplari:

```
#include <cmath>
```

float log(float X);

double log(double X);

long double log(long double X)

log() funksiyasi float tipi uchun logf(), long double tipi uchun esa logl()funksiyasini chaqiradi.

log10() – o'nli logarifm. Funksiya prototiplari:

#include <cmath>

float log10(float X);

double log10(double X);

long double log10(long double X);

log10() funksiyasi float tipi log10f(), long double tipi uchun log10l() funksiyasini chaqiradi.

fmod() – bo'linmaning qoldig'i. Funksiya prototiplari:

#include <cmath>

float fmod(float X, float Y);

double fmod(double X, double Y);

long double fmod(long double X, long double Y);

fmod() funksiyasi float tipi fmodf(), long double tipi uchun fmodl() funksiyasini chaqiradi. Funksiyadan foydalanishga misol:

cout << fmod(100.0, 9.0) << endl; *// 1*

cout << fmod(100.0, 10.0) << endl; *// 0*

modf() – haqiqiy sonini butun va kasr qismlarga ajratadi. Funksiya qiymat sifatida kasr qismni qaytaradi. Butun qismi ikkinchi parametrda berilgan o'zgaruvchida saqlanadi. Funksiya prototiplari:

#include <cmath>

float modf(float X, float *Y);

double modf(double X, double *Y);

long double modf(long double X, long double *Y) ;

modf() funksiyasi float tipidagi funksiya uchun modff() funksiyasini chaqiradi, long double tipi uchun – modfl() funksiyasini. Funksiyadan foydalanishga misol:

```
double x = 0.0;
cout << modf(12.5,&x) << endl; // 0.5
cout << x << endl; // 12
```

div() – ikki maydonli struktura qaytaradi: quot (X/Y butun bo'lish natijasi), rem (X%Y bo'linmaning qoldiq qismi). Funksiyaning prototipi:

```
#include <cstdlib>
div_t div(int X, int Y);
ldiv_t div(long X, long Y) ;
lldiv_t div(long long X, long long Y) ;
```

div() funksiyasi long tipi uchun ldiv() funksiyasini chaqiradi, long long tipi uchun – lldiv(). Funksiyadan foydalanishga misol:

```
div_t d;
d = div(13,2);
cout << d.quot << endl; // 6== 13/2
cout << d.rem << endl; // 1== 13/2
```

Sonlarni yaxlitlash uchun quyidagi funksiyalar tayinlangan:

ceil() – eng yaqin katta qiymatgacha yaxlitlangan qiymat qaytaradi. Funksiya prototiplari:

```
#include <cmath>
float ceil(float X);
double ceil(double X);
long double ceil(long double X);
```

ceil()funksiyasi float tipi uchun ceilf() funksiyasini chaqiradi, long double tipi uchun esa – ceill().

Funksiyadan foydalanishga misol:

```
cout << ceil(1.49) << endl; // 2
cout << ceil(1.5) << endl; // 2
cout << ceil(1.51) << endl; // 2
```

floor() – eng yaqin kichik qiymatgacha yaxlitlangan qiymat. Funksiya prototiplari:

```
#include <cmath>
float floor(float X);
```

double floor(double X);
long double floor(long double X);

floor() funksiyasi float tipi uchun floorf() funksiyasini chaqiradi, long double tipi uchun esa – floorl(). Funksiyadan foydalanishga misol:

```
cout << floor(1.49) << endl; // 1
cout << floor(1.5) << endl; // 1
cout << floor(1.51) << endl; // 1
```

3.4. Tasodifiy sonlarni generatsiyalash

Tasodifiy sonlarni generatsiyalash uchun foydalaniladigan funksiyalar:

rand() – '0' dan RAND_MAX gacha tasodifiy sonni generatsiyalaydi. Funksiyaning prototipi va RAND_MAX makrosining aniqlanishi:

```
#include<cstdlib>
int rand(void);
#define RAND_MAX 0x7fff
```

rand() funksiyasidan foydalanishga misol:

```
cout << rand() << endl; // 41
cout << rand() << endl; // 18467
cout << RAND_MAX << endl; // 32767
```

'0'dan ma'lum bir aniqlikdagi qiymatgacha bo'lgan, rand_max gacha emas, tasodifiy sonni olish uchun bo'linmaning qoldiqini olish operatoridan "%" foydalaniladi. Masalan: '0'dan '10'gacha sonlarni olishga misol:

```
cout << rand() % 11 << endl;
```

srand() – tasodifiy sonlar generatorini yangi ketma-ketlikka sozlaydi. Odatda parametr sifatida nolinchi ko'rsatkichli, 1970-yil 1-yanvardan boshlab o'tgan sekundlar sonini qaytaruvchi, time() funksiyasidan foydalaniladi.

```
#include <cstdlib>
void srand(unsigned int Seed);
#include <ctime>
```

time_t time(time_t *Time);

Misol:

```
srand(static_cast<unsigned int> (time(0)));
cout << rand() << endl;
```

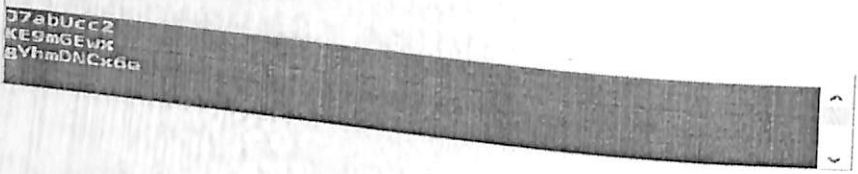
Agar srand() funksiyasi ayni bir parametr bilan chaqirilgan bo'lsa, u holda har bir chaqiruvda ayni bir tasodifiy son generatsiya qilinadi:

```
srand(100);
cout << rand() << endl; // 365
srand(100);
cout << rand() << endl; // 365
```

Misol tariqasida ihtiyoriy uzunlikdagi parollar generatorini yaratamiz. Buning uchun massiviga mumkin bo'lgan barcha belgilar qo'shiladi. So'ngra siklda tasodifiy element olinadi. Massiv elementini olgan belgini, birinchi element manzili birinchi parametr sifatida uzatilgan, yakuniy belgili massivga yoziladi. Belgili massiv oxiriga nol belgi qo'yiladi. Belgili massiv paroldagi belgilar sonidan hech bo'lmaganda bitta belgiga ko'p bo'lishi lozim.

```
// Parollarni generatsiya qiluvchi dastur
#include <iostream>
#include <cstdlib>
#include <ctime>
void passw_generator(char *pstr, int count_char);
int main() {
 char str[80];
 srand(static_cast<unsigned int>(time(0)));
 passw_generator(str, 8);
 cout << str << endl; // taxminan J7abUcc2
 passw_generator(str, 8);
 cout << str << endl; // KE9mGEwx
 passw_generator(str, 10);
 cout << str << endl; // gYhmDNCx6e
```

```
    cin.get();
    return 0;
}
void passw_generator(char *pstr, int count_char) {
    const short SIZE = 62;
    char mas[SIZE] = {'a','b','c','d','e','f','g','h','i',
'j','k','l','m','n','o','p','q','r',
's','t','u','v','w','x','y','z','A',
'B','C','D','E','F','G','H','I','J',
'K','L','M','N','O','P','Q','R','S',
'T','U','V','W','X','Y','Z','1','2',
'3','4','5','6','7','8','9','0' };
    for(int i = 0; i < count_char; ++i) {
    *pstr = mas[rand() % SIZE];
    ++pstr;
    }
    *pstr = '\0';
}
```



4. Amallar va oqimlar

4.1. Matematik va bitli amallar

Amallar berilganlar bilan ma'lum bir amallarni bajarish imkonini beradi. Masalan, o'zlashtirish amallari berilganlarni o'zgaruvchida saqlash uchun xizmat qiladi. Matematik amallar arifmetik amallarni bajarish uchun mo'ljallangan, shart amallari esa mantiqiy ifodaning qiymatiga bog'liq ravishda dasturning alohida bir qismini bajarishi yoki aksincha bajarmasligi mumkin. Quyida keltirilgan amallar arifmetik hisoblashlarni bajarish imkonini beradi:

+ – qo'shish: cout << 10 + 15 << endl; *// 25*

– – ayirish: cout << 35 – 15 << endl; *// 20*

- – unar minus: int x = 10; cout << -x << endl; // *-10*

* – ko'paytirish: cout << 25 * 2 << endl; // *50*

/ – butun bo'lish.

Butun sonlarni bo'lish amalga oshirilayotgan bo'lsa, qoldiq qismi "tashlab yuboriladi" va butun son qaytariladi. Haqiqiy sonlarni bo'lish klassik usulda amalga oshiriladi. Agar ifodada haqiqiy va butun son ishtirok etsa, u holda butun son avtomatik ravishda haqiqiy songa almashtiriladi. Masalan:

cout << 10 / 3 << endl; // *3*

cout << 10.0 / 3.0 << endl; // *3.33333*

cout << 10.0 / 3 << endl; // *3.33333*

% – bo'linmaning qoldiq qismi. Bu amalni haqiqiy sonlarga qo'llab bo'lmaydi. Masalan:

cout << 10 % 2 << endl; // *0 (10 – 10/2*2)*

cout << 10 % 3 << endl; // *1 (10 – 10/3*3)*

cout << 10 % 4 << endl; // *2 (10 – 10/4*4)*

cout << 10 % 6 << endl; // *4 (10 – 10/6*6)*

++ – inkrement amali. O'zgaruvchining qiymatini '1'ga oshiradi:

int x = 10;

++x; // x = x + *1; ga ekvivalent*

cout << x << endl; // *11*

-- – dekrement amali. O'zgaruvchining qiymatini '1'ga kamaytiradi:

int x = 10;

--x; // x = x – *1; ga ekvivalent*

cout << x << endl; // *9*

Inkrement va dekrement amallaridan postfiksli yoki *perefiksli* shaklda foydalanish mumkin:

x++; x--; // *Postfiksli shakl*

++x; --x; // *Perefiksli shakl*

Postfiksli shaklda (x++) o'zgaruvchining qiymati amaldan avval qaytariladi, perefiksli shaklda (++x) esa avval amal bajariladi va so'ngra qiymat qaytariladi. Buni misolda ko'ramiz.

```
using namespace std;
#include <iostream>
#include <clocale>
int main() {
 int x = 0, y = 0;
 /* setlocale(LC_ALL, "Russian_Russia.1251"); - Rus alif-
bosini tasvirlash
 x = 5; y = x++; // y = 5, x = 6
 cout << «Postfiksli shakl(y = x++;):» << endl;
 cout << «y = « << y << endl << «x = « << x << endl;
 x = 5; y = ++x; // y = 6, x = 6
 cout << «Perefiksli shakl(y = ++x;):» << endl;
 cout << «y = « << y << endl << «x = « << x << endl;
 cin.get();
 return 0;
}
```

```
Postfiksli shakl(y = x++;):
y = 5
x = 6
Perefiksli shakl(y = ++x;):
y = 6
x = 6
```

Agar inkrement va dekrement amallaridan murakkab ifodalarda foydalanilayotgan bo'lsa, ifodaning bajarilishi natijasi qanday bo'lisini tushunish qiyinlashadi. Masalan, quyidagi ko'rsatmalar bajarilganidan so'ng, "y" o'zgaruvchining qiymati qanday bo'ladi?

```
int x = 5, y = 0;
y=++x + ++x;
```

Bu ko'rsatmalar bajarilganidan so'ng o'zgaruvchi '14' (y = 7 + 7;) sonini oladi. Avval birinch ++x ifoda hisoblanadi. "x" o'zgaruvchisining qiymati '6(5+1)'ga teng bo'ladi. So'ngra ++x ikkinch ifoda. "x" o'zgaruvchisining qiymati '7(6+1)'ga teng bo'ladi. Natijada ko'rsatma quyidagicha ko'rinish oladi: x = 7; y = x + x;

"x" o'zgaruvchi '7' qiymat olgani uchun '7+7' ifoda hisoblanadi va natija '14' soni bo'ladi. Boshqa kompilyatordan foydalanilganida bagarilish natijasi bo'shqacha bo'lishi mumkin, chunki C++ tili standartida kompilyatorning holati (o'zini tutishi) aniqlanmagan.

Dasturni tahlil qilishni biladigan dasturchilar va boshqalar uchun murakkabliklar tug'dirmaslik uchun inkrement va dekrement amallaridan alohida perefiksli shaklda foydalangan ma'qul.

```
int x = 5, y = 0;
++x; ++x;
y = x + x;
```

Bunday ko'rsatmalar bajarilganidan so'ng, «y» o'zgaruvchining qiymati hech qanday ikkilanishlarni yuzaga keltirmaydi.

Bitli amallar alohida bitlarni boshqarish uchun mo'ljallangan. Bu operatorlarni haqiqiy sonlarga, mantiqiy qiymatlarga va boshqa murakkab tiplarga qo'llab bo'lmaydi. VC++ tili quyidagi bitli amallarni ta'minlaydi:

~ – ikkilik inversiya

Har bir bitning qiymati qaramaqarshisiga almashtiriladi:

```
char x = 100; // 01100100
x = ~x; // 10011011
```

| – ikkilik YOKI

```
char x = 100; // 01100100
char y = 75; // 01001011
char z = x|y; // 01101111
```

& – ikkilik VA

```
char x = 100; // 01100100
char y = 75; // 01001011
char z = x & y; // 01000000
```

^ – ikkilk istesnoli YOKI

```
unsigned char x = 100; //
01100100
unsigned char y = 250; //
11111010
unsigned char z = x ^ y; //
10011110
```

<< – chapga surish – ikkilikdagi sonni chaga bir va undan ortiq razryadga suradi va o'ngdagi razryadlarni nollar bilan to'ldiradi:

```
char x = 100; // 01100100
x = x << 1; // 11001000
x = x << 1; // 10010000
x = x << 2; // 01000000
```

>> – o'ngga surish – ikkilikdagi sonni o'ngga bir va undan ortiq razryadga suradi va chapdagi razryadlarni, agar son musbat bo'lsa, nollar bilan to'ldiradi. Son manfiy bo'lsa, razryadlar chapdan birlar bilan to'ldiriladi:

```
char x = 100; // 01100100
x = x >> 1; // 00110010
x = x >> 1; // 00011001
x = x >> 2; // 00000110
```

```
char x = -127; // 10000001
x = x >> 1; // 11000000
x = x >> 1; // 11100000
x = x >> 2; // 11111000
```

4.2. O'zlashtirish amallari

O'zlashtirish amallari qiymatlarni o'zgaruvchilarda saqlash uchun mo'ljallangan. C++ tilida mavjud o'zlashtirish amallarini keltirib o'tamiz.

= – o'zgaruvchiga qiymat o'zlashtiradi. Matematik tenglik belgisiga o'xshasa ham, C++ tilida uning ma'nosi umuman boshqacha. O'zlashtirish amalidan o'ng tomonda o'zgarmas yoki murakkab ifoda joylashishi mumkin. Amaldan chap tomonda o'zgarmas emas, balki o'zgaruvchi, ko'rsatkich yoki obyekt ("=" operatori yuklangan) joylashishi mumkin. Qiymatni o'zlashtirishga misol:

```
int x; x = 10;
x = 12 * 10 + 45 / 5;
12 + 45 = 45 + 5; // Bunday mumkin emas!
```

Bir ko'rsatmada bir necha o'zgaruvchga qiymat o'zlashtirish ham mumkin. Masalan: x = y = z = 2;

+= – o'zgaruvchining qiymatini ko'rsatilgan kattalikka orttiradi:

```
x += 10; // Ekvivalenti: x = x + 10;
```

–= – o'zgaruvchining qiymatini ko'rsatilgan kattalikka kamaytiradi:

```
x –= 10; // Ekvivalenti: x = x – 10;
```

*= – o'zgaruvchining qiymatini ko'rsatilgan kattalikka ko'paytiradi:

```
x *= 10 + 5; // Ekvivalenti: x = x * (10 + 5);
```

/= – o'zgaruvchining qiymatini ko'rsatilgan kattalikka butun bo'ladi:

x /= 2; *// Ekvivalenti:* x = x / 2;

%= – o'zgaruvchining qiymatini ko'rsatilgan kattalikka bo'ladi va qoldiqni qaytaradi:

x %= 2; *// Ekvivalenti:* x = x %2 ;

&=, |= , ^= , <<= va >>= – o'zlashtirish bilan bitli amallar.

4.3. Taqqoslash amallari

Taqqoslash amallari mantiqiy ifodalarda foydalaniladi. C++ tilida mumkin bo'lgan taqqoslash amallarini sanab o'tamiz:

== – teng; != – teng emas;

> – katta; < – kichik;

>= – katta yoki teng; <= – kichik yoki teng.

Mantiqiy ifodalar faqat ikkita qiymat qaytaradi: true (rost, '1' soniga mos keladi) yoki false (yolg'on, '0' soniga mos keladi). Mantiqiy ifodaning qiymatini chiqarishga doir misollar:

```
cout << (10 == 10) << endl; // 1(true)
cout << (10 == 5) << endl; // 0 (false;
cout << (10 != 5) << endl; // 1 (true)
cout << (10 < 5) << endl; // 0 (false;
cout << (10 > 5) << endl; // 1 (true)
```

Mantiqiy ifoda taqqoslash amallaridan birortasini ham olmasligi mumkin. Bu holda '0' soni avtomatik mantiqiy qiymatga (false), har qanday nol bo'lmagan qiymat (manfiy qiymat ham) haqiqiy qiymatga (true) almashadi:

```
cout << (bool)10 << endl; // 1 (true)
cout << (bool)-10 << endl; // 1 (true)
cout << (bool)12.5 << endl; // 1 (true)
cout << (bool)0.1 << endl; // 1 (true)
cout << (bool)0.0 << endl; // 0 (false)
cout << (bool)0 << endl; // 0 (false)
```

Bu misollarda bool tipiga keltirishdan foydalanildi. Mantiqiy ifoda ichida tiplarga keltirishni bajarish shart emas, u

avtomatik ravishda amalga oshiriladi. Shuni e'tiborga olish lozimki, tenglikka tekshirish amali ikkita "=" belgisini oladi. Bitta "=" (*o'zgaruvchiga qiymat berish*) belgisini ko'rsatish mantiqiy xatolik hisoblanadi. O'zlashtirish amalidan mantiqiy ifoda ichida ishlatishga ruxsat beriladi, shu sababli kompilyator xatolik haqida xabar bermaydi. Ammo, dastur noto'g'ri bajarilishi mumkin. Dasturlashni boshlayotganlar bunday xatolikka yo'l qo'yadilar. Masalan, quyidagi misolda '11' soniga tenglikni tekshirish o'rniga o'zlashtirish amali bajariladi:

```
int x = 10;
if(x = 11) {
 cout << "x == 11" << endl;
}
```

Har qanday '0'ga teng bo'lmagan son true mantiqiy qiymat sifatida qabul qilinadi, shu sababli rost bo'lgan true == true shartini tekshirish amalga oshiriladi.

Mantiqiy ifodaning qiymatini "!" amali yordamida invertatsiya qilish (almashtirish) mumkin. Bu holda agar mantiqiy qiymat false qaytarsa, u holda !false – true qiymat qaytaradi. Masalan:

```
cout << (10 == 5) << endl; // 0 (false)
cout << (10 != 5) << endl; // 1 (true)
```

&& – mantiqiy VA. Mantiqiy qiymat ikkala ifoda ostlari true bo'lgandagina true qiymat qaytaradi. Masalan:

```
cout << ((10 == 10) && (5 != 3)) << endl; // 1 (true)
cout << ((10 == 10) && (5 == 3)) << endl; // 0 (false)
```

|| – matiqiy YOKI. Ikkita tasdiqdan birortasi true qiymat qaytarsa, mantiqiy ifoda ham true qaytaradi. Shuni hisobga olish lozimki, agar ifoda osti true qiymat qaytarsa, navbatdagi ifoda osti hisoblanmaydi. Masalan, f1()||f2()||f3() mantiqiy ifodada f2() funksiya f1() funksiya false qiymat qaytargandagina chaqiriladi, f3() funksiya esa agar f1() va f2() funksiyalar false qiymat qaytarsagina chaqiriladi.

```
cout << ((10 == 10) || (5 != 3)) << endl; // 1 (true)
cout << ((10 == 10) || (5 == 3)) << endl; // 1 (true)
```

&& va || amallari yordamida bir necha mantiqiy amalni katta bir guruhga birlashtirish mumkin. && va || amallarining bajarilish natijalari quyidagi jadvalda keltirilgan:

a	b	a&& b	a|b
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

4.4. Amallarning bajarilish ustunliklari

Barcha amallar ustunlik darajasi bo'yicha bajariladi. Avval eng yudori darajagali amalga ega bo'lgan ifoda hisoblanadi. Masalan, ko'paytirishli ifoda qo'shish amalli ifodadan avva bajariladi, chunki ko'paytirish amalining ustunlik darajasi qo'shishga nisbatan yuqori. Agar amallarning ustunlik darajasi teng bo'lsa, aniq amal uchun aniqlangan hisoblash tartibidan foydalaniladi. O'zlashtirish va unar amallari o'ngdan chapga bajariladi. Matematik, bitli va taqqoslash amallari, hamda vergul amali chapdan o'ngga bajariladi.

Ifodani hisoblash ketma-ketligini qavslar yordamida o'zgartirish mumkin.

```
int x = 0;
x = 5 + 10 * 3 / 2; // ko'paytirish -> bo'lish -> qo'shish
cout << x << endl; // 20
x = (5 + 10) * 3 / 2; // qo'shish -> ko'paytirish -> bo'lish
cout << x << endl; // 22
```

Amallarning (operandlarning) ustunlik darajalari

Ustunlik	Amallar
1(yuqori)	( ) , [ ] , • , - >
2	++, —, -, !, *(qayta yuklash), &(manzilni olish), –(unar minus)
3	.*, ->*
4	*, /, %
5	+, –(ayirish)
6	<<, >>
7	<, >, <=, >=
8	==, !=
9	&(bitli VA)
10	^
11	|
12	&&
13	||
14	=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=
15(quyi)	,(vergul)

4.5. O‘qish-yozish oqimlari

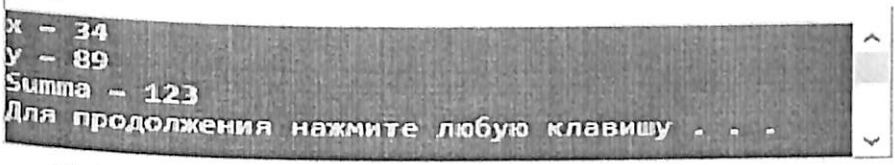
VC++ muhitida berilganlarni kiritish uchun iostream faylida e’lon qilingan cin obyekti tayinlangan. cin obyekti biriktirilgan ihtiyoriy tipdagi berilganlarni kiritish imkonini beradi, masalan, son, belgi yoki satr.

cin obyektidan foydalanishdan avval include ko‘rsatmasi yordamida iostream faylini ulash zarur: #include <iostream>

Quyidagi misolda cin obyektidan foydalanishga misol tariqasida foydalanuvchi tomonidan kiritilgan ikkita sonni qo‘shish listinggi keltirilgan.

```
#include <iostream>
int main() {
 int x = 0, y = 0;
```

```
cout << "x = "; cin >> x;
cout << "y = "; cin >> y;
cout << "Summa = " << x + y << endl;
return 0;
}
```



Birinchi satrda iostream faylini ulash amalga oshiriladi. Keyin main() funksiyasining ichida ikkita lokal o'zgaruvchi e'lon qilinadi: "x" va "y". Satr boshidagi int kalit so'zi butun o'zgaruvchilar e'lon qilinayotganligini anglatadi. E'lon qilishda o'zgaruvchilarga "=" operatori yordamida '0' boshlang'ich qiymat yuklanadi. Agar qiymat yuklanmasa, "**musor**" deb nomlanuvchi ihtiyoriy qiymat oladi. Bir satrda, vergul bilan ajratgan holda, bir necha o'zgaruvchini e'lon qilish mumkin.

Navbatdagi satrda cout obyekti yordamida foydalanuvchiga ("x=") ko'rsatma chiqariladi. Bu ko'rsatma asosida foydalanuvchi nima talab qilinayotganligini bilib oladi. cin obyekti yordamida son kiritilib, [Enter] klavishi bosilganidan keyin, qiymat "x=" o'zgaruvchiga o'zlashtiriladi. Bunda cin obyektidan keyin kiritish yo'nalishini ko'rsatuvchi ">>" operatori va undan keyin kiritilgan ma'lumotlar saqlanadigan o'zgaruvchi nomi beriladi. cout va cin obyektlari std nomlar fazosida e'lon qilinganligi sababli, obyektlardan oldin nomlar fazosi nomini ko'rsatish zarur.

Keyin "y" o'zgaruvchida saqlanadigan ikkinchi sonni qabul qilish haqida navbatdagi ko'rsatma chiqariladi va yuqoridagi amallar takrorlanadi. Navbatdagi satrda ikki sonni qo'shish va natijani chiqarish amalga oshiriladi.

Bu jarayonlar quyidagicha ko'rinishga ega bo'ladi:

x = 10

y = 20

Summa = 30

Berilganlarni kiritishda o'zgaruvchi tipiga almashtirishga harakat amalga oshiriladi. Bizning misolda '10' va '20' satrlari mos ravishda '10' va '20' sonlariga almashtiriladi. Ammo, foydalanuvchi son o'rniga belgi kiritishi mumkin. Bunday hollarda almashtirish amalga oshirilmaydi va o'zgaruvchiga qiymat o'zlashtirilmaydi. Kiritilgan qiymat esa buferda qoladi va avtomatik ravishda navbatdagi kiritish amali tomonidan o'qiladi.

Almashtirish jarayonida cin obyektining good() metodi yordamida xatolik yo'qligini tekshirish mumkin. Metodning prototipi quyidagicha:

```
bool good() const;
```

good() metodi xatoliklar bo'lmasa, '1' soniga mos keluvchi true, aks holda '0' soniga mos keluvchi false mantiqiy qiymat qaytaradi. Xatolik haqida xabar chiqarish va dasturning bajarilishini uzish quyidagicha bajarilishi mumkin:

```
cout << "x = ";
cin >> x;
if(cin.good() == false) {
 cout << "Xato" << endl;
 return 1; // main() funksiyasidan chiqish
}
```

Keltirib o'tilgan misolda shartni tekshirish uchun if tarmoqlanish operatonidan foydalanilgan. Operatordan so'ng, qavs ichida tekshiriluvchi ifoda ko'rsatiladi. Agar ifoda mantiqiy true (rost) qiymati qaytarsa, figurali qavs ichidagi ko'rsatmalar bajariladi. Agarda false (yolg'on) qiymati qaytariladigan bo'lsa, u holda boshqaruv yopuvchi qavsdan keyin turgan ko'rsatmaga uzatiladi. good() metodi qaytaradigan qiymatni tekshirish "==" operatori yordamida bajariladi. Xatolik bo'lganda good() metodi false qiymat qaytaradi (false == false) va xatolik haqida xabar chiqarilib, '1' qitmat qaytargan holda main() funksiyasidan chiqiladi.

Tenglikka tekshirishda bitta "=" belgisini ko'rsatish mantiqan noto'g'ri hisoblanadi, chunki bu operatordan o'zgaruvchiga qi-

ymat o'zlashtirish uchun foydalaniladi. Mantiqiy ifoda ichida o'zlashtirish operatoridan foydalanish mumkin. Shu sababli kompilyator xatolik haqida xabar bermaydi, ammo dastur noto'g'ri bajarilishi mumkin. Masalan, quyidagi misolda '11' soniga tenglikka tekshirish o'rniga o'zlashtirish amali bajariladi.

```
int x = 10;
if(x = 11) {
 cout << "x == 11" << endl;
}
```

Har qanday "0" bo'lmagan qiymat mantiqiy true qiymat sifatida qabul qilinadi. Shu sababli, rost bo'lgan "true == true" shartni tekshirish amalga oshiriladi. Shartning true qiymatga moslikni tekshirish ko'rsatilmaganda (по умолчанию) amalga oshiriladi.

VC++ muhitida berilganlarni chiqarish uchun iostream faylida e'lon qilingan cout, cerr va clog obyektlaridan foydalaniladi. cout obyektidan oddiy xabarlarni konsol oynasiga chiqarishda foydalaniladi. cerr va clog obyektlari xatoliklar haqidagi xabarlarni chiqarish uchun qo'llaniladi. cout obyekti kabi cerr va clog obyektlari birinchi navbatda konsol oynasi bilan bog'langan, ammo oqimni boshqa qurilmaga yoki faylga burishi mumkin. Obyektlarning barcha imkoniyatlarini tushunish uchun obyektga yo'naltirilgan dasturlash prinsiplarini bilish talab qilinadi. Biz cout obyektining asosiy imkoniyatlari bilan tanishib chiqamiz.

Obyektlardan foydalanishdan avval include ko'rsatmasi yordamida iostream faylini ulash lozim:

```
#include <iostream>
```

Fayl nomi burchakli qavslarda "h" kengaytmasiz ko'rsatiladi, chunki iostream fayli C++ning standart kutubxonasi tarkibiga kiradi. Fayl ulanganidan so'ng barcha obyektlar std nomlar fazosi orqali kirishli bo'ladi. Obyektga murojaat qilishda nomlar fazosini ko'rsatish va obyekt nomidan oldin ikkita ikki nuqta (::) belgisini qo'yish zarur. Masalan, "Salom, Olam!" satrini quyidagicha chiqarish mumkin:

std::cout << "Salom, Olam!";

Har safar nomlar fazosini ko'rsatishga erinsangiz, u holda barcha identifikatorlarni ko'rsatma yordamida global nomlar fazosiga import qilish mumkin: using namespace std;

Bunday holda nomlar fazosi nomi ko'rsatilmaydi:

cout << "Salom, Olam!";

Bu juda qulay bo'lsa ham, barcha identifikatorlarni global nomlar fazosiga import qilishdan saqlanish lozim, bu nomlarning to'qnashuvini keltirib chiqarishi mumkin. Bu usul o'rniga alohida identifikatorlarni import qilishdan foydalangan yaxshiroq. Masalan, faqat cout obyektini quyidagicha import qilish mumkin: using cout;

Obyekt nomidan keyin chiqarish yo'nalishini ko'rsatuvchi "<<" operator ko'rsatiladi. Foydalanuvchi obyektlarini Ishlab chiqishda bu operatorni yuklash va shu orqali chiqarishni boshqarish mumkin. "<<" operatoridan keyin konsol oynasida ko'rinadigan obyekt beriladi. Obyekt sifatida son, satr, hamda "<<" operatori yuklangan obyektni ko'rsatish mumkin. Misol:

```
cout << 10; // butun son
cout << 81.5; // haqiqiy son
cout << "Salom, Olam!"; // satr
char str[] = "Salom, Olam!";
cout << str; // satr
```

Bu misolning bajarilishi natijasi quyidagicha bo'ladi:

1081.5Salom, Olam!Salom, Olam!

Natijadan ko'rinib turibdiki, barcha berilganlar bitta satrda chiqariladi. Alohida satrlarda chiqarish uchun endl ("end line"ning qisqartmasi) o'zgarmasidan foydalanish mumkin. Masalan:

```
cout << "String1";
cout << endl;
cout << "String2";
```

Konsol oynasida quyidagicha natija chiqariladi:

String1
String2

Aslida endl o'zgarmas emas. Aytish mumkinki, bu ichida belgini chiqarish, satrni ko'chirishni bajariladigan va chiqish oqimiga murojjat qaytaruvchi funksiya (*monipulyator* deb nomlanadi). Murojaatni qaytarishi hisobiga "<<" operatorlaridan zanjir qurish mumkin. Masalan, yuqoridaa kletirilgan misolni quyidagicha ko'rinishda yozish ham mumkin:

cout << "String1" << endl << "String2";

Satrni ko'chirish uchun satrni ko'chirish belgisini ifodalovchi '\n' belgilar kombinatsiyasidan ham foydalanish mumkin.

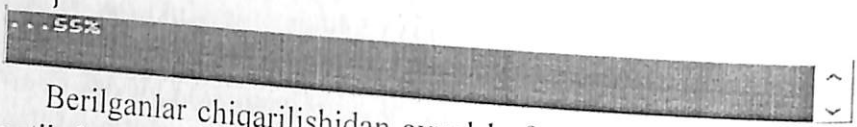
Masalan, cout << "String1" <<'\n'<< "String2";

Ikki baytli belgilardan tashkil topgan satrlarni chiqarish uchun iostream faylida e'lon qilingan wcout, wcerr va wclog obyektlari tayinlangan. wcout obyekti oddiy xabarlarni, wcerr va wclog obyektlari xatoliklar haqidagi xabarlarni chiqarish uchun mo'ljallangan. Misol: wcout << L"String";

Standart chiqarish yordamida konsol oynasida jarayonning bajarilishi indikatorini yaratish mumkin. Bunday indikatorni amalga oshirish uchun Windowsda satrni ko'chirish belgisi ikkita belgidan tashkil topganligini eslash zarur: '\r' (karetkani ko'chirish) va '\n' (satrni ko'shirish). Faqat bitta karetkani ko'chirish '\r' belgisidan foydalangan holda satrning boshiga ko'chish va avval chiqarilgan ma'lumotni qayta yozish mumkin. Quyida jarayon indikatoriga doir misol listinggi keltirilgan.

```
#include <iostream>
int main() {
 int i, j;
 cout << "... 0%";
 for(i = 5; i < 101; i += 5) {
 for(j = 0; j < 500000000; ++j); // jarayonning imitatsiyasi
 cout << "\r..." << i <<"%";
 }
```

```
cout << endl;
return 0;
}
```



Berilganlar chiqarilishidan avval buferga joylashtirilishi va natijada buferni tozalash talab qilinishi mumkin. Buferni tozalash flush() metodi yordamida amalga oshiriladi. Metodning ko'rinishi: cout.flush();

4.6. Belgilarni interfaol kiritish

cin obyekti belgini [Enter] klavishi bosilgandan keyin olish imkonini beradi. Agar klavish bosilgandan keyin darhol belgini olish zarur bo'lsa, u holda conio.h faylida e'lon qilingan _getche() va _getch() funksiyalaridan foydalanish mumkin. Funksiyalar prototiplari:

```
#include <conio.h>
int_getche(void);
int_getch(void);
```

Ba'zi kompilyatiorlarda _getche() va _getch() funksiyalari mos ravishda getche() va getch() deb nomlanadi.

_getche() funksiyasi belgi kodini qaytaradi va uni ekranga chiqaradi. Bunda faqat ingliz tili alifbosi harflari to'g'ri tasvirlanadi.rus harflari kiritilganda konsol oynasida harflar noto'g'ri tasvirlanishi mumkin. Klaviturадan biror klavish bosilganda _getch() funksiyasi belgi kodini qaytaradi, belgi ekranga chiqarilmaydi. Bunday holat funksiyadan yashirin ma'lumotlarni (masalan, parolni) olish uchun foydalanish imkonini beradi. Bunda, ba'zi bir xizmatchi klavishlar bosilganida ham belgi kodi qaytarilishini e'tiborga olish lozim, masalan, [Home], [End] va boshqalar.

_getch() funksiyasidan foydalanishga misol tariqasida parolni kiritishning to'g'riligini tekshiruvchi dastur yaratamiz. Parol '16' tagacha belgi olishi mumkin ('0' dan '9' gacha raqamlar va

ihtiyoriy registrdagi "a" dav "z" gacha lotin harflari). Klavish bosilganda ekranga yulduzcha chiqariladi. Agar belgi mumkin bo'lgan belgilar to'plamiga kirmasa, xatolik haqida xabar chiqariladi va dastur ishini yakunlaydi. Kiritilgan parolni tekshirish [Enter] klavishi bosilgandan keyin amalga oshiriladi.

```
#include <iostream>
#include <conio.h> // _getch() uchun
#include <cstring> // strcmp() uchun
int main() {
char passwd[17], ch;
bool flag = false; int i = 0;
cout << "Password:";
do {
ch = _getch();
if(i > 15||ch == '\r'||ch == '\n') {
flag = true;
passwd[i] = '\0';
} else if((ch > 47 && ch < 58) // "0" dan "9" gacha raqamlar
||(ch > 64 && ch < 91) // "A" dan "Z" gacha harflar
||(ch > 96 && ch < 123)) { // "a" dan "z" gacha harflar
passwd[i] = ch;
cout << '*';
++i;
} else { // Mumkin bo'lmagan belgi bo'lsa, chiqamiz
cout << endl << "Xato" << endl;
return 0;
}
} while(!flag);
if(strcmp(passwd, "test") == 0) { // Maydonlarni taqqoslash
cout << endl << "OK" << endl;
} else {
cout << endl << "Xato" << endl;
```

```
}
return 0;
}
```

```
password:**********
xato
Для продолжения нажмите любую клавишу . . .
```

Dasturning boshida iostream, conio.h va cstring fayllari qo‘shiladi. iostream fayli cout obyektidan foydalanish uchun kerak, conio.h fayli – _getch() funksiyasi uchun, cstring fayli – ikkita satrni taqqoslash uchun mo‘ljallangan strcmp() funksiyasi uchun. cout obyekti va strcmp() funksiyasi std nomlar fazosi ichida, _getch() funksiyasi global nomlar fazosida joylashgan. Shu sababli, _getch() funksiyasidan oldin nomlar fazosi ko‘rsatilmaydi. main() funksiyasi ichida to‘rtta o‘zgaruvchi e’lon qilinadi. ‘17’ta elementi olgan (16 belgi +’\0’) passwd belgili massivga kiritilgan belgilar yoziladi. Belgili o‘zgaruvchi ‘ch’ga bosilgan klavish kodi yoziladi. flag mantiqiy o‘zgaruvchisi kiritish statusini saqlash uchun mo‘ljallangan (agar true qiymat olsa, parolni kiritish tugallangan), butun tipli “i” o‘zgaruvchisi passwd massivi ichida joriy indeksni saqlash uchun tayinlangan. Bu yerda bir narsaga e’tibor bering, massivning birinchi indeksi ‘1’ emas, ‘0’ qiymat qabul qiladi.

O‘zgaruvchilar e’lon qilinganidan so‘ng, cout obyekti yordamida foydalanuvchiga yordam chiqatiladi. do...while sikli ichida getch() funksiyasi yordamida joriy belgi olinadi va uning kodi ch o‘zgaruvchisida saqlanadi. Keyingi satrda indeksning massiv chegarasidan chiqishi (i>15) va [Enter] klavishining bosilishi tekshiriladi. [Enter] bosilganda \r\n (karetkani va satrni ko‘chirish) ketma-ketlik uzatiladi. Ammo, _getch() funksiyasi faqat bitta belgi olishi mumkin, shu sababli ‘\r’ yoki ‘\n’ belgining mavjudligi tekshiriladi. Mantiqiy ifodalar o‘zaro || (mantiqiy YOKI) operatori orqali ajratiladi. Butun ifoda “rost” qiymat qaytarishi uchun mantiqiy ifodalardan birortasining “rost” bo‘lishi yetarli.

Agar i>15 ifoda yolg'on bo'lsa, u holda ch=='\r' ifoda tekshiriladi. Bu ifoda rost bo'lsa, butun ifoda rost hisoblanadi va keyingi tekshirish amalga oshirilmaydi. Agar ch=='\r' ifoda yolg'on bo'lsa, u holda ch=='\n' ifoda tekshiriladi. ch=='\n' ifoda yolg'on bo'lsa butun ifoda "yolg'on", aks holda "rost" hisoblanadi. Agar ifoda "rost" bo'lsa, parolni kiritish tugallandi. Sikldan chiqish uchun flag o'zgaruvchisiga true qiymati o'zlashtiriladi. Bundan tashqari, massivning oxiriga nol belgi qo'yiladi. U kiritilgan parol satrining oxirini bildiladi. Bu ko'rsatmalardan keyin boshqaruv sikl oxiriga uzatiladi va flag shrti tekshiriladi. Shart false (!true) qaytarganda sikldan chiqish amalga oshiriladi. So'ngra strcmp() funksiyasi yordamida parolning to'g'riligi tekshirilib, mos xabar chiqariladi. strcmp() funksiyasi parametr sifatida ikkita satrni oladi va satrlar teng bo'lsa, '0' sonini qaytaradi. Taqqoslash belgilar registrlarni hisobga olgan holda bajariladi, chunki bu yerda satrdagi belgilar emas, ularning manzillarini taqqoslash amalga oshiriladi ("==" operatori orqali ikkita satrni taqqoslash mumkin emas).

Agar parolni kiritish tugallanmagan bo'lsa, belgining mumkin bo'lgan diapozonga ('0' dan '9' gacha raqamlar va ihtiyoriy registrdagi 'a' dan 'z' gacha lotin harflari) tegishliligi tekshiriladi. Bunday tekshiruvni amalga oshirish zarur, chunki _getch() funksiyasi ba'zi bir xizmatchi klavishlarning kodini ham qaytaradi, masalan, [Home], [End] va boshqalar. Agar shart bajarilmasa, xatolik haqida xabar chiqariladi va dasturning bajarilishi tugallangan holda '0' qiymat qaytariladi.

Tekshiriladigan shart qavslar ichida bir necha kichik qismlarga bo'linadi. Birinchi qavsda belgining '0' dan '9' gacha bo'lgan raqamlar oralig'iga mos kelishi tekshiriladi. '0' raqami '48' kodiga mos keladi, '9' raqami esa –'57' kodiga. Boshqa raqamlarning kodlarii bu kodlar oralig'ida yotadi. Ifodada barcha raqamlarning kodini ko'rsatib o'tmaslik uchun &&. (mantiqiy VA) operatori bilan ajratilgan ikkita shartni tekshirishdan foydalani-

ladi. (ch>47 && ch<58) ifoda quyidagicha o'qiladi: "agar ch o'zgaruvchi '47' dan katta va '58'dan kichik kodga ega bo'lgan belgini olsa, u holda true qiymat qaytaradi, aks holda – false qiymat". Birinchi qavs ichidagi shart rost bo'lsa, butun ifoda rost hisoblanadi va keyingi tekshirishlar bajarilmaydi. Shart yolg'on bo'lsa, navbatdagi qavs ichidagi shart tekshiriladi. Uchinchi qavs ichidagi shart faqat ikkinchi qavs ichidagi shart yolg'on bo'lgandagina tekshiriladi. Belgilar kodlari o'rniga apostroflar ichida belgilarning o'zini ko'rsatish mumkin. Bu holda shart quyidagicha ko'rinishga ega bo'ladi:

else if((ch >= '0' && ch <= '9') // *"0" dan "9" gacha raqamlar*

||(ch >= 'A' && ch <= 'Z') // *"A" dan "Z" gacha harflar*

||(ch >= 'a' && ch <= 'z')) // *"a" dan "z" gacha harflar*

Agar belgi mumkin bo'lgan oraliqda bo'lsa, massivda saqlanadi va konsol oynasiga yulduzcha belgisi chiqariladi. So'ngra "i" o'zgaruvchida qiymat bittaga orttirilib, ko'rsatkich massivning navbatdagi elementiga ko'chiriladi. ++i ifodasi i=i+1 ifodaga mos keladi. Bu ko'rsatmalardan keyin boshqaruv siklning oxiriga uzatiladi va !flag shartni trkshirish amalga oshiriladi. Shart true (!false) qiymat qaytarsa, sikl ichidagi ko'rsatmalar yana bir bor bajariladi.

5. Operatorlar

5.1. Tarmoqlanuvchi operatorlar

if tarmoqlanuvchi operatori. if tarmoqlanish operatori mantiqiy ifodaning qiymatiga bog'liq holda dasturning alohida blokini bajarishi yoki uni bajarmasligi mumkin. Operator quyidagicha formatga ega:

```
if(<mantiqiy ifoda>) {
 <Shart rost bo'lsa, bajariluvchi blok>
} else {
 <Shart yolg'on bo'lsa, bajariluvchi blok>
 }
```

Agar *mantiqiy ifoda* true (rost) qiymat qaytarsa, u holda if operatoridan keyin joylashgan figurali qavs ichidagi ko'rsatmalar bajariladi. Agar mantiqiy ifoda false (yolg'on) qiymat qaytarsa, u holda else kalit so'zidan keyin joylashgan figurali qavs ichidagi ko'rsatmalar bajariladi. else bloki majburiy hisoblanmaydi. Agarda blok bitta ko'rsatmadan iborat bo'lsa, figurali qavslarni ko'rsatish shart emas.

Misol tariqasida foydalanuvchi tomonidan kiritilgan sonni juftlikka tekshirishni va unga mos xabarni chiqarishni ko'raylik.

```
#include <iostream>
#include <clocale>
int main() {
 int x = 0;
 cout << "Son kiriting: ";
 cin >> x;
 if(!cin.good()) {
 cout << endl << "Siz son kiritmadingiz" << endl;
 cin.clear(); // Xato bayroqchasini tashlash
 cin.ignore(255,'\n').get();
 } else {
 if(x % 2 == 0)
 cout << x << " – juft son" << endl;
 else cout << x << " – toq son" << endl;
 cin.ignore(255,'\n').get();
 }
 return 0;
}
```

```
Son kiriting: 789
789 - toq son
```

Misoldan ko'rinib turibdiki, bir tarmoqlanish operatorini ikkinchisi ichiga joylash mumkin. Birinchi if operatori sonni kiritishda xatolikning yo'qligini tekshiradi. good() metodi xatoliklar mavjud bo'lmasa, true mantiqiy qiymatini yoki aks hol-

da false qiymatini qaytaradi. E'tibor bersangiz, mantiqiy ifoda taqqoslash operatorlarini olmagan (invertatsiya "!" operatoridan tashqari): if(!cin.good()) {

Shatrning true qiymatiga mosligini tekshirish ko'rsatilmaganda amalga oshiriladi. Xatolik bo'lganda good() metodi false qiymat qaytaradi. Bundan kelib chiqadiki, !false==true sharti rost bo'ladi. Bu mantiqiy ifodani quyidagicha yozish mumkin edi:

```
if(cin.good() == false) { yoki quyidagicha:
if(cin.good() != true) {
```

Bu hola bunday tekshirishlar ortiqcha hisoblanadi. Funksiyaning nomini ko'rsatish va zarurat yuzaga kelganda qiymatni inversiyalash yetarli. Son kiritilganda xatoliklar bo'lmasa, else bloki bajariladi.

Agar sonni kiritishda xatolik bo'lmasa, blok bajariadi. Bu blokda sonni juftlikka tekshirivchi ichki tarmoqlanuvchu operator joylashgan. %2==0 shartga bog'liq holda mos xabar chiqariladi. Son ikkiga qoldiqsiz bo'linsa, % operatori "0", aks holda "1" sonini qaytaradi.

```
if(x % 2 == 0)
 cout << x << " – juft son" << endl;
 else cout << x << " – toq son" << endl;
cin.ignore(255, '\n').get();
```

E'tibor bering, tarmoqlanuvchi operator figurali qavslarni olmagan. Bu holda blokning ichida faqat bitta ko'rsatma bor. Oxirgi ko'rsatma else blokiga tegishli emas. U shartga bog'liq bo'lmagan holda bajariladi.

Bloklarni bir tekisda yozish noqulayliklarni yuzaga keltiradi, ya'ni ularni tushunish qiyinlashadi. Su sababli, har doim blok ichidagi ko'rsatmalarni bir xil chekinishda joylashtirish lozim. Chekinish sifatida tabulyatsiya yoki bir necha (bo'shliq) probeldan foydalanish mumkin. Ichki bloklarni yozishda probellar soni ichki darajaga ko'paytiriladi. Masala, birinchi darajali blok-

da 3 ta probel bo'sa, ikkinchi darajada '6'ta, uchunchi darajada '9'ta probel va h.k. Agar blok bir necha ko'rsatmalardan tashkil topgan bo'lsa, u holda figurali qavslarni ko'rsatish lozim.

```
if(<Shart1>) {
// Ko'rsatmalar
} else {
 if(<Shart2>) {
// Ko'rsatmalar
 } else {
// Ko'rsatmalar
 }
 }
```

Bir necha shartni tekshirish uchun bu sxemani o'zgartirish mumkin:

```
if(<Shart1>) {
// <Blok 1>
} else if(<Shart2>) {
// <Blok 2>
 } //...Fragment tushurib qoldirilgan...
 else if(<ShartN>) {
// <Blok N>
 } else {
// <else bloki>
 }
```

Agar <Shart 1> rost bo'lsa, u holda <Blok1> bajariladi va boshqa barcha shartlar tushirib qoldiriladi. Agar <Shart1> yolg'on bo'lsa, u holda <Shart2> tekshiriladi. <Shart2> rost bo'lsa, u holda <Blok2> bajariladi va boshqa barcha shartlar tushirib qoldiriladi. Agar barcha shartlar yolg'on bo'lsa, <else bloki> bajariladi. Misol tariqasida foydalanuvchi '0'dan '2'gacha sonlarning qaysi birini kiritganligini aniqlaymiz.

```
#include <iostream>
#include <clocale>
```

```
int main() {
 int x = 0;
 cout << "0 dan 2 gacha bo'lgan son kiriting: ";
 cin >> x;
 if(!cin.good()) {
 cout << endl << "Siz son kiritmadingiz" << endl;
 cin.clear(); //Xatolik bayroqchasini tashlash
 }
 else if(x == 0)
 cout << "Siz 0 sonini kiritdingiz" << endl;
 else if(x == 1)
 cout << "Siz 1 sonini kiritdingiz" << endl;
 else if(x == 2)
 cout << "Siz 2 sonini kiritdingiz" << endl;
 else {
 cout << "Siz boshqa son kiritdingiz" << endl;
 cout << "x = " << x << endl;
 }
 cin.ignore(255, '\n').get();
 return 0;
}
```

```
0 dan 2 gacha bo'lgan son kiriting: 9
Siz boshqa son kiritdingiz
x = 9
```

? : operatori (ifodasi). Shartni tekshirish uchun if operatori o'rniga ?: operatoridan ham foydalanish mumkin. Operatorning formati quyidagicha:

<O'zgaruvchi> = <mantiqiy ifoda> ? <Rost bo'lgandagi ifoda> : <Yolg'on bo'lgandagi ifoda>;

Agar mantiqiy ifoda true qiymat qaytarsa, so'roq belgisidan keyingi joylashga ifoda, false qiymat qaytarsa, u holda ikki nuqta belgisidan keyin joylashga ifoda bajariladi. Ifodalardan birortasining bajarilishi operator bajarilishining natijasi bo'ladi.

Masalan, sonni juftlikka tekshirish va natijani chiqarish quyidagicha ifodalanadi:

```
int x = 10;
cout << x;
((x % 2 == 0) ? " – jufa son" : " – toq son");
```

E'tibor bering operandlar sifatida ko'rsatmalar emas, aynan ifodalar ko'rsatilladi. Bundan tashqari, ifodalar albatta qanaydir qiymat qaytarishi lozim. Operator qiymat qaytargani uchun undan ifodalar ichida ham foydalanish mumkin.

```
int x, y; x = 0;
y = 30 + 10 / (! x ? 1 : x); // 30 + 10 / 1
cout << y << endl; // 40
x = 2;
y = 30 + 10 / (! x ? 1 : x); // 30 + 10 / 2
cout << y << endl; // 35
```

Operand sifatida qiymat qaytaruvchi funksiyani (*keyinroq o'rganiladi*) ham ko'rsatish mumkin.

```
int func1(int x) {
 cout << x <<" – juft son"<< endl;
 return 0;
}
int func2(int x) {
 cout << x <<" – toq son"<< endl;
 return 0;
}
// ... Dastur qismi tushurib qoldirilgan. ...
int x = 10;
(x % 2 == 0) ? func1(x) : func2(x);
```

switch operatorining (tanlashning) formati quyidagicha ko'rinishda:

```
switch(<Ifoda>) {
 case <Konstanta_1>: <Ko'rsatmalar>; break;
 case <Konstanta_2>: <Ko'rsatmalar>; break;
```

...
case <Konstanta_N>: <Ko'rsatmalar>; break;]
[default: <Ko'rsatmalar>]
}

switch operatori shart o'rniga ifoda qabul qiladi. Ifodaning qiymatiga bog'liq holda, bu qiymat ko'rsatilgan case bloklaridan biri bajariladi. Ifodaning qiymati butun son bo'lishi lozim. Agar hech bir qiymat case bloklarida bayon qilinmagan bo'lsa, u holda default (agar ko'rsatilgan bo'lsa) bloki bajariladi. case blokidagi o'zgarmaslar bir xil qiymat qabul qilishi mumkin emas.

switch operatoridan foydalanishga misol:

```
#include <iostream>
#include <clocale>
int main() {
 int OS = 0;
 cout << "Siz qanday operatsion tizimdan foydalanasiz?\n";
 cout << "1 - Windows XP\n";
 cout << "2 - Windows Vista\n";
 cout << "3 - Windows 7\n";
 cout << "4 - Boshqasi\n\n";
 cout << "Javobga mos keluvchi sonni kiriting: ";
 cin >> OS;
 if(!cin.good()) {
 cout << endl << "Siz son kiritmadingiz" << endl;
 cin.clear(); // Xatolik bayrog'ini tashlash
 cin.ignore(255, '\n').get();
 return 0;
 }
 cout << endl;
 switch(OS) {
 case 1: cout << "Sizning tanlovingiz-Windows XP"; break;
 case 2: cout << "Sizning tanlovingiz-Windows Vista";
break;
```

```
    case 3: cout << "Sizning tanlovingiz-Windows 7"; break;
    case 4: cout << "Sizning tanlovingiz–Boshqasi"; break;
    default: cout << "Operatsion tizimingizni aniqlay olmadik";
    }
    cout << endl;
    cin.ignore(255, '\n').get();
    return 0;
   }
```

```
Siz qanday operatsion tizimdan foydalanasiz ?
1 - Windows XP
2 - Windows Vista
3 - Windows 7
4 - Boshqasi

Javobga mos keluvchi sonni kiriting: 3

Sizning tanlovingiz-Windows 7
```

Misoldan ko'rinib turibdiki, casening har bir bloki oxirida break operatori ko'rsatilgan. Bu operator switch tanlash operatoridan mudatidan oldin chiqish imkonini beradi. Agar break operatori ko'rsatilmasa, ko'rsatilgan qiymatdan qat'iy nazar, casening navbatdagi bloki bajariladi.

Ba'zi hollarda bu foydali bo'lishi mumkin. Masalan, ko'rsatmalarni qiymatlar diapozonining oxiriga joylashtirib, turli qiymatlarda ayni bir ko'rsatmalarni bajarish mumkin. Misol:

```
switch (ch) {
 case 'a':
 case 'b':
 case 'c': cout << "a, b yoki c"; break;
 case 'd': cout << "Faqat d";
}
```

Sanaluvchi tip. Sanab o'tish – o'zgaruvchining mumkin bo'lgan barcha qiymatlarini ifodalovchi butun o'zgarmaslar mosligidir. Agar o'zgaruvchiga sanab o'tilgan qiymatlar bilan mos kelmaydigan qiymat o'zlashtirilsa, kompilyator xatolik

haqida xabar beradi. Sanab o'tishni e'lon qilish quyidagicha formatga ega:

enum[<Sanash nomi>] {
<O'zgarmaslar ro'yxati (vergul bilan ajratilgan>
} [<O'zgaruvchilarni e'lon qilish (vergul bilan)>];

O'zgaruvchining e'loni yopuvchi figurali qavsdan keyin yoki sanaluvchi tipdan berilganlar tipi sifatida foydalanib, alohida beriladi:

[enum]<Sanash nomi><O'zgaruvchilar nomlari>;

Sanaluvchi tipni e'lon qilishda enum kalit so'zini ko'rsatmaslik mumkin.

Sanaluvchi tip va o'zgaruvchini bir vaqtda e'lon qilishga misol:

enum Color {
red, blue, green, black
} color1;

bunda, red, blue, green va black o'zgarmaslarga avtomatik ravishda, noldan boshlab, butun qiymatlar o'zlashtiriladi. Tartiblash chapdan o'ngga amalga oshiriladi. Demak, red o'zgarmasi '0' qiymat oladi, blue – 1, green – 2, black esa – 3. O'zgaruvchini alohida e'lon qilishga misol keltiramiz:

Color color2;

Sanaluvchi tipni e'lon qilisha o'zgarmasga qiymat berish mumkin. Bunda navbatdagi o'zgarmas, bu o'zgarmasdan bittaga katta qiymat oladi. Masalan:

enum Color {
red = 3, blue, green = 7, black
} color1;

Misolda red o'zgsaruvchisi '0' emas, '3' qiymatga ega, blue – '4', green – '7', black esa – '8'.

O'zgaruvchiga quyidagicha qiymat berish ham mumkin:

color1 = black;

Sanaluvchi tip nomi va o'zgaruvchilar e'lonini bir vaqtda berish shart emas. Bunday hollarda figurali qavs ichida ko'rsatilgan o'zgaruvchilardan o'zgarmas sifatida foydalaniladi.

```
enum {
 red, blue = 10, green, black
} ;
cout << blue << endl; // 10
```

Sanaluvchi tipdan foydalanishga misol listinggi.

```
#include <iostream>
enum Color { // Sanaluvchi tipni e'lon qilish
 red, blue, green = 5, black
};
int main() {
 Color color; // O'zgaruvchini e'lon qilish
 color = black;
 cout << color << endl; // 6
 if(color == black) { // Qiymatni tekshirish
 cout << "color = black" << endl;
 }
 cin.get();
 return 0;
}
```

```
6
color = black
```

5.2. Takrorlash operatorlari

for takrorlash operatori. Sikl operatorlari ayni bir ko'rsatmalarni ko'p bor bajarish imkonini beradi. Faraz qilaylik 1 dan 100 gacha sonlarni satrda bittadan chiqarish zarur. Oddiy yo'l bilan 100 ta satr kodi yozish zarur bo'ladi:

```
cout << 1 << endl;
cout << 2 << endl;
cout << 100 << endl;
```

Sikl yordamida bu amallarni bitta satr kodi yordamida bajarish mumkin:

```
for(int i = 1; i <= 100; i++)
```

```
cout << i << endl;
```

for siklidan ifodalarni ma'lum bir sonda bajarish uchun foydalaniladi. Sikl quyidagicha formatga ega:

```
for(<Boshlang'ich qiymat>; <Shart>; <Orttirma>) {
 <Ko'rsatmalar>
}
```

<Boshlang'ich qiymat> – o'zgaruvchi-hisoblagichga boshlang'ich qiymat o'zlashtiradi, <Shart> – mantiqiy ifoda oladi. Mantiqiy ifoda true qiymat qaytarganda, sikl ichidagi ko'rsatmalar bajariladi, <Orttirma> – har bir iteratsiyada (takrorlanish) o'zgaruvchi-hisoblagichning o'zgarishini beradi.

for sikli ishining ketma ketligi quyidagicha:

1. O'zgaruvchi-hisoblagichga boshlang'ich qiymat beriladi.
2. Shart tekshiriladi, agar u rost bo'lsa sikl ichidagi ifodalar bajariladi, aks holda siklning bajarilishi tugallanadi.
3. O'zgaruvchi-hisoblagich <Orttirma>da ko'rsatilgan kattalikka o'zgaradi.
4. 2-punktga qaytadi.

O'zgaruvchi-hisoblagich sikldan tashqarida yoki <Boshlang'ich qiymat> parametrida e'lon qilinishi mumkin. Agan u parametrda e'lon qilingan bo'lsa, faqat sikl ichida ko'rinadi. Bundan tashqari, o'zgaruvchini sikldan tshqarida e'lon qilish va unga boshlang'ich qiymat o'zlashtirishga ham ruxsat beriladi. Bunday hollarda <Boshlang'ich qiymat> parametrini bo'sh qoldirish mumkin. Misol ko'raylik.

```
int i; // Sikldan tashqaridagi e'lon
for(i = 1; i <= 20; i++) {
 cout << i << endl;
} // 'i' o'zgaruvchi sikldan tashqarida ko'rinadi
cout << i << endl;
// Sikl ichida e'lon qilish
for(int j = 1; j <= 20; j++) {
```

```
cout << j << endl;
} // 'j' o'zgaruvchi sikldan tashqarida ko'rinmaydi
int k = 1; // Sikldan tashqarida nomlash
for(; k <= 20; k++) {
 cout << k << endl;
}
```

Sikl <Shart> false qiymat qaytarmaguncha bajariladi. Agar bu amalga oshirilmasa, sikl cheksiz bo'ladi. <Shart> parametrida ko'rsatilgan mantiqiy ifoda har bir iteratsiyada hisoblanadi. Shu sababli, agar mantiqiy ifoda ichida qandaydir hisoblashlar bajarilsa va sikl ichida qiymat o'zgarmasa, u holda hisoblashni <Boshlang'ich qiymat> parametriga chiqarish lozim. Bunday hollarda hisoblash o'zgaruvchi-hisoblagichga qiymat o'zlashtirilganidan keyin, orqali ko'rsatiladi. Masalan,

```
for(int i = 1, j = 10 + 30; i <= j; i++) {
 cout << i << endl;
}
```

<Orttirma> parametrida ko'rsatilgan ifoda o'zgaruvchi-hisoblagich qiymatini nafaqat orttirishi, balki kamaytirishi, bundan tashqari, orttirma qiymat har qanday kattalikka o'zgarishi mumkin.

```
for(int i = 100; i > 0; i--) { // 100 dan 1 gacha sonlarni chiqaramiz
 cout << i << endl;
}
 // 1 dan 100 gacha juft sonlarni chiqaramiz
for(int j = 2; j <= 100; j += 2) {
 cout << j << endl;
}
```

Agar ozgaruvchi-hisoblagich sikl ichida o'zgarsa, u hoda <Orttirma> parametridagi ifodani umuman ko'rsatmaslik ham mumkin.

```
for(int i = 1; i <= 10;) {
```

```
cout << i << endl;
i++; // Orttirma
}
```

for siklining barcha barametrlari va sikl ichidagi ko'rsatmalar majburiy hisoblanmaydi. Parametrlarni ko'rsatish shart bo'lmasa ham, nuqtali vergul bo'lishi shart. Barcha parametrlar ko'rsatilmasa, sikl cheksiz bo'ladi. Cheksiz sikldan chiqish uchun break operatoridan foydalanish lozim.

```
#include <clocale>
int i = 1; // <Boshlang'ich qiymat>
for(; ;) { // Cheksiz sikl
 if(i <= 10) { // <Shart>
 cout << i << endl;
 i++; // <Orttirma>
 } else {
 break; } // Sikldan chiqamiz
}
```

while takrorlash operatori. while siklida ifodalarning bajarilishi mantiqiy ifoda rost bo'ganda davom etadi. Sikl quyidagicha formatga ega:

```
<Boshlang'ich qiymat>;
while <Shart>) {
 <Ko'rsatmalar>;
 <Orttirma>;
}
```

while siklining bajarilish ketma-ketligi:

1. O‘zgaruvchi-hisoblagichga boshlang‘ich qiymat o‘zlashtiriladi.

2. Shart tekshiriladi, agar u rost bo‘lsa sikl ichidagi ko‘rsatmalar bajariladi, aks holda siklning bajarilishi tugallanadi.

3. O‘zgaruvchi-hisoblagich <Orttirma>da ko‘rsatilgan kattalikka o‘zgaradi.

4. 2-punktga o‘tiladi.

while siklidan foydalangan hola 1 dan 100 gacha bo‘lgan barcha sonlarni chiqaramiz:

```
int i = 1; // <Boshlang'ich qiymat>
while(i <= 100) { // <Shart>
 cout << i << endl; // <Ko'rsatmalar>
 ++i; // <Orttirma>
}
```

Agar <Orttirma> ko‘rsatilmasa, sikl cheksiz bo‘ladi.

do ... while takrorlash operatori. do...while siklida ifodalarning bajarilishi mantiqiy ifoda rost bo‘ganda davom etadi. while siklidan farqli ravishda shart sikl boshida emas, oxirida tekshiriladi. Shi sabsbli, do...while ichidagi ko‘rsatmalar hech bo‘lmaganda bir marta bajariladi. Sikl quyidagicha formatga ega:

```
<Boshlang'ich qiymat>;
do {
 <Ko'rsatmalar>;
 <Orttirma>;
} while(<Shart>);
```

do...while siklining bajarilish ketma-ketligi:

1. O‘zgaruvchi-hisoblagichga boshlang‘ich qiymat o‘zlashtiriladi.

2. Sikl ichidagi ko‘rsatmalar bajariladi.

3. O‘zgaruvchi-hisoblagich <Orttirma>dagi kattalikka o‘zgaradi.

4. Shart tekshiriladi, agar u rost bo'lsa, 2-punktga o'tiladi, aks holda siklning bajarilishi tugallanadi.

do...while siklidan foydalangan hola 1 dan 100 gacha bo'lgan barcha sonlarni chiqaramiz:

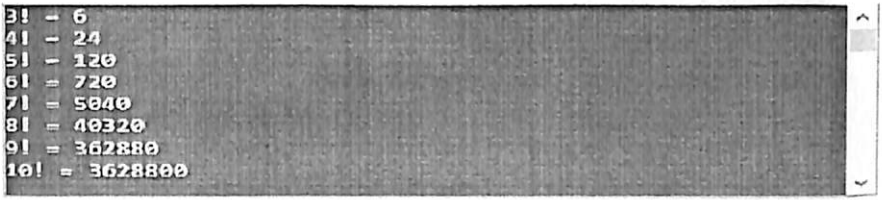
```
int i = 1; // <Boshlang'ich qiymat>
do {
cout << "i = " << i << endl; // <Ko'rsatmalar>
++i; // <Orttirma>
} while(i <= 100); // <shart>
```

Agar <Orttirma> ko'rsatilmasa, sikl cheksiz bo'ladi.

Rekursiya. Rekursiya – bu funksiyaning o'zini o'zi chaqirish imkoniyati. Funksiyaning har bir chaqirilishida lokal o'zgaruvchilarning yangi to'plami yaratiladi. Rekursiyadan avvaldan noaniq tarkibga ega bo'lgan obyektni saralash, amallarning noma'lum sonda bajarilishi uchun foydalanish qulay. Rekursiyani qo'llashning tipik misoli faktorialni hisoblash hisoblanadi.

```
#include <iostream>
unsigned long long factorial(unsigned long n);
int main() { // Faktorialni hisoblash dasturi
 for(int i = 3; i < 11; ++i) {
 cout << i << "! = " << factorial(i) << endl;
 }
 cin.get();
 return 0;
}
unsigned long long factorial(unsigned long n) {
if(n <= 1)
 return 1;
 else return n * factorial(n-1);
}
```

Dasturning bajarilishi natijasi:



5.3. Boshqaruvni uzatish operatorlari

continue, break va goto operatorlari. continue operatori sikl ichidagi barcha ifodalarning bajarilishi tugallanmasdan navbatdagi iteratsiyaga o'tish imkonini beradi. Misol tariqasida, 5 dan 10 gacha bo'lgan sonlardan tashqari, 1 dan 100 gacha sonlarning barchasini chiqarish dasturini ko'raylik:

```
for(int i = 1; i <= 100; ++i) {
 if(i > 4 && i < 11) continue;
 cout << i << endl;
}
```

break operatori siklning bajarilishini uzish imkonini beradi. Misol tariqasida 1 dan 100 gacha sonlarning barchasini yana bir usul bilan chiqaramiz:

```
int i=1;
while(1) {
 if(i > 100) break;
 cout << i << endl;
 ++i;
}
```

Bu yerda shartda '1' qiymati ko'rsatildi. Bunda sikl ichidagi ko'rsatmalar cheksiz bajariladi. Ammo, break operatoridan foydalanish, '100'ta satr chop qilinganidan keyin, uning bajarilishini uzadi. break operatori dasturning emas, balki siklning bajarilishini uzadi, ya'ni boshqaruv sikldan keyin keltirilgan ko'rsatmaga o'tadi. Cheksiz sikldan break operatori bilan birgalikda foydalanish foydalanuvchi tomonidan avvaldan aniqlanmagan sondagi ma'lumotlarni olishda qulay. Misol tariqasida noaniq sondagi butun sonlarni yig'ish dasturini ko'ramiz:

```
#include <iostream>
#include <clocale>
int main() {
 int x = 0, summa = 0;
 cout << "Natijani olish uchun 0 sonini kiriting" << endl;
 for(;;) {
 cout << "Son kiriting: ";
 cin >> x;
 if(!cin.good()) {
 cout <<"Siz son kiritmadingiz!"<<endl;
 cin.clear(); // Xatolik bayroqchasini tashlash
 cin.ignore(255,'\n');
 continue;
 }
 if(!x) break;
 summa += x;
 }
 cout << "Sonlar yig'indisi: " << summa <<endl;
 cin.ignore(255,'\n').get();
 return 0;
}
```

```
Natijani olish uchun 0 sonini kiriting
Son kiriting: 6
Son kiriting: 45
Son kiriting: -76
Son kiriting: 781
Son kiriting: 0
Sonlar yig'indisi: 756
```

goto shartsiz o'tish operatori orqali boshqaruvni dasturning ihiyoriy qismiga o'tkazish mumkin. Operator quyidagicha formatga ega:

goto <Nishon>;

<Nishon> parametridagi qiymat mumkin bo'lgan identifikator bo'lishi lozim. Dasturning boshqaruv uzatilayotgan joyi bir o'lchamli nison bilan belgilanadi va undan so'ng ikki nuqta ":" ko'rsatiladi. Misol tariqasia '1' dan '100'gacha sonlarni chiqarishni ko'ramiz:

```
int i = 1;
BLOCK_START: {
if(i > 100) goto BLOCK_END;
cout << i << endl;
++i;
goto BLOCK_START;
}
BLOCK_END:;
```

Demak, figurali qavslardan nafaqat shartli, siklik operatorlarda, funksiyalarda, balki alohida konstruksiya sifatida ham foydalanish mumkin. Figurali qavslar ichiga olingan kod fragmenti ***blok*** deyiladi. Blok ichida e'lon qilingan o'zgaruvchilar faqat blok doirasida ko'rinadi.

goto operatoridan imkon qadar foydalanmaslikka intilish zarur, chunki u dasturni juda chalkash qiladi va kutilmagan natijalarga olib kelishi mumkin.

6. Statik massivlar

Massiv – bir tipga tegishli bo'lgan o'zgaruvchilarning tartiblangan (nomerlangan) to'plami. Massivdagi o'zgaruvchi *element* deyiladi, uning massivdagi o'rni *indeks* orqali beriladi. Massivning barcha elementlari xotiraning qo'shni katakchalarida joylashadi. Masalan, long int (4 bayt band qiladi) tipidagi uchta elementdan iborat massiv e'lon qilingan bo'lsin. U holda massiv birinchi elementining manzili (32 bitli operatsion tizimda) 0x0041A040, ikkinchisiniki – 0x0041A044, uchinchisiniki esa – 0x0041A048 bo'ladi. Massiv band qilgan xotira hajmi (baytlarda) quyidagicha aniqlanadi:

<Xotira hajmi> = sizeof(<Tip>) *<Elementlar soni>

6.1. Bir o'lchovli massivlar

Massiv quyidagicha e'lon qilinadi:

<Tip><O'zgaruvchi>[<Elementlar soni>];

Masalan, uch elementdan tashkil topgan long int tipidagi massivni e'lon qilish quyidagicha bo'ladi: long mas[3];

E'lon qilishda massiv elementlariga boshlang'ich qiymatlar berish mumkin. Buning uchun massiv e'lon qilinganidan so'ng "=" operatori ko'rsatiladi va figurali qavslar ichida qiymatlar beriladi. Qiymatlar bir-biri biri bilan vergul bilan ajratiladi. Yopiluvchi figurali qavsdan keyin nuqtali vergulni ko'rsatish majburiy. Massivni nomlashga misol:

long mas[3] = {10, 20, 30};

Figurali qavslar ichidagi qiymatlar massiv elementlari sonidan kam bo'lishi mumkin. Bu hoda elementlarga qiymatlar massiv boshidan boshlab o'zlashtiriladi. Masalan: long mas[3] = {10, 20};

Massivning birinchi elementiga '10', ikkinchi elementiga '20' va uchunchi elementiga '0' qiymati o'zlashtiriladi.

Massivning barcha elementlariga '0' qiymatini o'zlashtirish quyidagicha bo'ladi: int mas[15] = {0};

Massiv e'lonida boshlang'ich qiymatlar ko'rsatilayotgan bo'lsa, kvadrat qavs ichida elementlar sonini ko'rsatmaslik mumkin. Massivning o'lchami figurali qavs ichidagi elementlar soniga mos keltiriladi. Misol:

long mas[] = {10, 20, 30};

Massivni e'lon qilish jarayonida boshlang'ich qiymatlar ko'rsatilmasa, global massiv elementlariga avromatik ravishda '0' o'zlashtiriladi, lokal massiv elementlari «musor» deb nomlanuvchi ihtiyoriy qiymat oladi.

Massiv elementlariga murojaat, element indeksi ko'rsatiluvchi, kvadrat qavslar orqali amalga oshiriladi. Massiv elementlarini tartiblash '0'dan boshlanadi. Indekslar yordamida massiv elementlariga, e'lon qilinganidan keyin, boshlang'ich qiymat berish mumkin:

```
long mas[3];
mas[0] = 10; // Birinchi element "0" indeksga ega !!!
arr[1] = 20; // Ikkinchi element
mas[2] = 30; // Uchinchi element
```

Kompilyatsiya jarayonida indeksning qiymatlar diapozoni-

dan chiqib ketishi tekshirilmaydi. Bunda qiymatni xotiraning qo'shni yacheykasiga yozish va dasturning ishchi qobiliyatini buzish, hattoki operatsion tizimni ishdan chiqarish, mumkin. Indekslar to'g'riligini nazorat qilish dasturchining vazifasiga kiradi.

Massiv aniqlanganidan so'ng, zarur bo'lgan xotira ajratiladi va o'zgaruvchida massiv birinchi elementining manzili saqlanadi. Kvadrat qavslar ichida indeks ko'rsatilganda, massiv elementining mos manzilini hisoblash amalga oshiriladi. Massiv elelemntining manzilini bilgan holda uning qiymatini olish yoki qayta yozish mumkin. Boshqacha so'z bilan aytadigan bo'lsak, massiv elementlari ustida oddiy o'zgaruvchilardagi kabi amallarni bajarish mumkin. Misol:

```
long mas[3] = {10, 20, 30};
long x = 0;
x = mas[1] + 12;
mas[2] = x – mas[2];
cout << x << endl; // 32
cout << mas[2] << endl; // 2
```

6.2. Massivlar bilan ishlash

Massivni saralash – bu elementlarni o'sish yoki kamayish tartibida tartiblash. Saralash natijalarni chiqarishda, hamda qiymatlarni qidirishni tayyorlashda qo'llaniladi. Saralangan massiv bo'yicha qiymatni qidirish, saralanmaganga nisbatan, ancha tez amalga oshiriladi. har safar massivning barcha elementlarini ko'rib chiqish shart emas.

Massiv elementlarini saralashda for operatoridan foydalangan qulay. Birinchi parametrda o'zgaruvchi-hisoblagichga '0' qiymat o'zlashtiriladi. Davom etish sharti massiv elementlari sonidan kam o'zgaruvchi-hisoblagich qiymati hisoblanadi. Uchinchi parametrda siklning har bir iteratsiyasida (qaytarilishida) birga ortish ko'rsatiladi. Sikl ichida elementlarga kirish ichida o'zgaruvchi-hisoblagich ko'rsatilgan kvadrat qavslar yor-

damida amalga oshiriladi. Misol tariqasida massivning barcha elementlarini nomerlaymiz, so'ngra barcha elementlarini to'g'ri va teskari tartibda chiqarishni ko'ramiz:

```
const short SIZE = 20;
int mas[SIZE];
// Massivning barcha elementlarini nomerlash
for(int i = 0, j = 1; i < SIZE; i++, j++) {
 mas[i] = j;
}
// Qiymatlar to'g'ri tartibda kiritiladi
for(int i = 0; i < SIZE; i++) {
 cout << mas[i] << endl;
}
cout << " " << endl;
// Qiymatlar teskari tartibda chiqariladi
for(int j = SIZE - 1; j >= 0; j--) {
 cout << mas[j] << endl;
}
```

Bu misolda massiv elementlari soni o'zgarmas sifatida e'lon qilingan (SIZE o'zgarmasi). Bu juda qulay, har bir saralashda massivning o'lchamini ko'rsatishga to'g'ri keladi. Agar elementlar soni son ko'rinishida berilgan bo'lsa, massiv o'lchami o'zgarganda barcha qiymatlarni qo'lda o'zgartirishga to'g'ri keladi. O'zgarmasdan foydalanilganda esa, elementlar sonini bir joyda o'zgartirish yetarli.

Massivning dasturdagi elementlari sonini olish uchun massivning baytdagi umumiy o'lchamini o'lcham tipiga bo'lish zarur. Bu o'lchamni sizeof operatori orqali olish mumkin. Masalan,

```
int mas[15] = {0};
cout << sizeof mas / sizeof(int) << endl; // 15
```

for siklini har doim while sikli bilan almashtirish mumkin. Misol tariqasida elementlarin teskari tartibda nomerlash va barcha qiymatlarni chiqarishni ko'raylik:

```
const short SIZE = 20;
int mas[SIZE];
// Barcha elementlarni nomerlash
int i = 0, j = SIZE;
while(i < SIZE) {
 mas[i] = j;
 ++i;
 --j;
}
// Massivning qiymatini chiqarish
i = 0;
while(i < SIZE) {
 cout << mas[i] << endl;
 ++i;
}
```

Massiv elementlarini tartiblash uchun ko'pincha *pufakchali saralash* (*пузырьковая сортировка*) deb nomlanuvchi usuldan foydalaniladi. Bu usulda eng kishik qiymat massivning boshlanishiga, eng katta qiymat esa massivning oxiriga o'tadi. Saralash bir necha o'tishda amalga oshiriladi. har bir o'tishda ketma-ket joylashgan ikkita qiymat taqqoslanadi. Agar birinchi element ikkinchisidan katta bo'lsa, elementlar qiymatlari o'rnini almashtiradi.

Besh belgidan iborat massivni saralash uchun maksumum to'rt marda o'tish va o'n marta taqqoslash zarur bo'ladi. Agar o'tishda birorta o'rin almashtirish bo'lmasa, saralashni uzish mumkin. Bunday holda avvaldan tartiblangan massivni tartiblash uchun bitta o'tish yetarli.

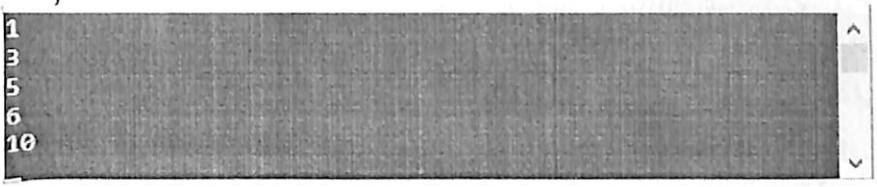
Pufakchali saralashda elementlarni taqqoslash ketma-ketligi

O'tish	*Bayoni*	*Massiv elementlarining qiymati*				
	Massiv element-lari	arg[0]	arg[1]	arg[2]	arg[3]	arg[4]
	Boshlang'ich qiymatlar	10	5	6	1	3
1	arg[3] va arg[4]	10	5	6	1	3
	arg[2] va arg[3]	10	5	1	6	3
	arg[1] va arg[2]	10	1	5	6	3
	arg[0] va arg[1]	1	10	5	6	3
2	arg[3] va arg[4]	1	10	5	3	6
	arg[2] va arg[3]	1	10	3	5	6
	arg[1] va arg[2]	1	3	10	5	6
3	arg[3] va arg[4]	1	3	10	5	6
	arg[2] va arg[3]	1	3	5	10	6
4	arg[3] va arg[4]	1	3	5	6	10

Qiymatlarning o'sishi bo'yicha pufakchali saralash dasturi listinggi.

```
#include <iostream>
int main() {
 const short SIZE = 5;
 int mas[SIZE] = {10, 5, 6, 1, 3};
 int j = 0, tmp = 0, k = SIZE – 2;
 bool is_swap = false;
 for(int i = 0; i <= k; i++) {
 is_swap = false;
 for(j = k; j >= i; j--) {
 if(mas[j] > mas[j + 1]) {
 tmp = mas[j + 1];
 mas[j + 1] = mas[j];
 mas[j] = tmp;
```

```
is_swap = true;
}
}
// O'rin almashtirishlar bo'lmasa chiqish
if(!is_swap) break;
}
// Saralangan massivni chiqarish
for(int i = 0; i < SIZE; i++) {
cout << mas[i] << endl;
}
cin.get();
return 0;
}
```



Navbatdagi misolda qiymatlarning kamayishi tartibida saralashni amalga oshiramiz. Dastur foydali bo'lishi uchun o'tish yo'nalishini o'zgartiramiz. Qiymatlarning kamayishi bo'yicha pufakchali saralash dasturi listingi.

```
#include <iostream>
int main() {
 const short SIZE = 5;
 int mas[SIZE] = {10, 5, 6, 1, 3};
 int j = 0, tmp = 0;
 bool is_swap = false;
 for(int i = SIZE – 1; i >= 1; i--) {
 is_swap = false;
 for(j = 0; j < i; ++j) {
 if(mas[j] < mas[j + 1]) {
 tmp = mas[j + 1];
 mas[j + 1] = mas[j];
```

```
mas[j] = tmp;
is_swap = true;
}
}
if(!is_swap) break;
}
for(int i = 0; i < SIZE; i++) {
cout << mas[i] << endl;
}
cin.get();
return 0;
}
```

Massivni saralash uchun standart qsort() funksiyasidan ham foydalanish mumkin. Funksiyaning umumiy ko'rinishi quyidagicha:

```
#include <cstdlib>
void qsort(void *Base, size_t NumOfElements,
size_t SizeOfElements, int(*PtFuncCompare)
(const void*, const void*));
```

Base parametrida massiv birinchi elementining manzili ko'rsatiladi, NumOfElements parametrida – massiv elementlari soni, SizeOfElements parametrida – har bir elementning o'lchami (baytda).

Ichida ikkita elementni taqqoslash amalga oshiriladigan foydalanuvchi funksiyasi manzili (funksiya nomi qavs va parametrlarsiz beriladi) oxirgi parametrda beriladi. Foydalanuvchining taqqoslash funksiyasi prototipi quyidagicha ko'rinishda bo'ladi:

```
int<Funksiya nomi>(const void *arg1, const void *arg2);
```

O‘sish bo‘yicha saralashda “arg1 < arg2” bo‘lsa, funksiya manfiy, “arg1 > arg2” bo‘lsa, musbat yoki qiymatlar teng bo‘lsa nol ‘0’ qiymat qaytarishi lozim. Funksiya ichida void* ko‘rsatkichini aniq bir tipga keltirishni bajarish zarur. qsort() funksiyasidan foydalanishga misol quyidagi listingda keltirilgan.

```
#include <iostream>
#include <cstdlib>
int mysort(const void *arg1, const void *arg2);
int main() {
 const short SIZE = 5; int mas[SIZE] = {10, 5, 6, 1, 3};
 qsort(mas, SIZE, sizeof(int), *mysort);
 for(int i = 0; i < SIZE; i++) {
 cout << mas[i] << endl;
 }
 cin.get();
 return 0;
}
int mysort(const void *arg1, const void *arg2) {
 return *(int*)arg1 - *(int*)arg2;
}
```

```
1
3
5
6
10
```

Kamayish tartibida saralashni amalga oshirish uchun qiymatlarning o‘rnini almashtirish yetarli:

```
int mysort(const void *arg1, const void *arg2) {
 return *(int*)arg2 – *(int*)arg1;
}
```

Minimal va maksimal qiymatni qidirish. Massivning eng kichik elementini topish uchun min o‘zgaruvchisiga massiv birinchi elementining qiymati o‘zlashtiriladi va boshqa ele-

mentlar bilan taqqoslanadi. Agar joriy elementning qiymati min o'zgaruvchisidagi qiymatdan kichik bo'lsa, bu qiymat min o'zgaruvchisiga yuklanadi. Maksimal qiymatni topish uchun max o'zgaruvchisiga massiv birinchi elementining qiymati o'zlashtiriladi va boshqa elementlar bilan taqqoslanadi. Joriy elementning qiymati max o'zgaruv-chisining qiymatidan katta bo'lsa, u max o'zgaruvchisiga yuklanadi.

Bir o'lchovli massivning minimal va maksimal qiymatlarini topishga misol quyidagi listingda keltirilgan.

```
#include <iostream>
int main() {
const short SIZE = 5; int mas[SIZE] = {2, 5, 6, 1, 3};
int min = mas[0], max = mas[0];
for(int i = 1; i < SIZE; i++) {
if(min > mas[i]) min = mas[i];
if(max < mas[i]) max = mas[i];
}
cout << "min = " << min << endl;
cout << "max = " << max << endl;
cin.get();
return 0;
}
```

```
min = 1
max = 6
```

Massivlarni taqqoslash uchun memcmp() funksiyasidan foydalaniladi. memcmp() funksiyasi Buf1 va Buf2 massivlarning birinchi Size baytlarini taqqoslaydi. Funksiya qiymat sifatida quyidagilarni qaytaradi:

- ✓ *manfiy son* – agar Buf1 < Buf2;
- ✓ *nol* – agar massivlar teng bo'lsa;
- ✓ *musbat son* – agar Buf1 > Buf2.

Funksiyaning prototipi:

```
#include <cstring>
```

int memcmp(const void *Buf1, const void *Buf2, size_t Size);

Misol:

```
int mas1[3] = {1, 2, 3}, mas2[3] = {1, 2, 3}, x = 0;
x = memcmp(mas1, mas2, sizeof mas2);
cout << x << endl;
mas1[2] = 2; // mas1[] = {1,2,2}, arr2[]
x = memcmp(mas1, mas2, sizeof mas2);
cout << x << endl;
mas1[2] = 4; // mas1[]={1,2,4}, mas2[]
x=memcmp(mas1, mas2, sizeof mas2);
cout << x << endl;
```

memcmp() funksiyasi belgilarning registrini hisobga olgan holda taqqoslashni amalga oshiradi. Agar belgilarni hisobga olmagan holda taqqoslash lozim bo'lsa, _memicmp() funksiyasidan foydalanish mumkin. Rus xarflari uchun lokalni sozlash zarur bo'ladi. Funksiyaning prototipi:

```
#include <cstring>
int _memicmp(const void *Buf1, const void *Buf2, size_t Size);
```

Parametrlarning tayinlanishi va qaytaruvchi qiymati memcmp() funksiyasi kabi. Funksiyadan foydalanishga misol:

```
setlocale(LC_ALL, "Russian_Russia.1251"); // Rus alifbosi uchun
char str1[] = "абв", str2[] = "АБВ"; int x = 0;
x = _memicmp(str1, str2, sizeof str2);
cout << x << endl; // 0
x = memcmp(str1, str2, sizeof str2);
cout << x << endl; // 1
```

6.3. Ko'p o'lchovli massivlar

Massivlar ko'p o'lchovli bo'lishi ham mumkin. Ko'p o'lchovli massivlarni e'lon qilish quyidagicha formatga ega:

<Tip><O'zgaruvchi>[<Elementlar soni1>]...[<Elementlar soniN>];

Amaliyotda asosan ma'lum bir satr va ustunlarni olgan jadval katakchalaridagi qiymatlarni saqlash imkonini beruvchi ikki o'lchovli massivlardan foydalaniladi.

Ikki o'lchovli massiv quyidagicha e'lon qilinadi:

<Tip><O'zgaruvchi>[<Satrlar soni>][<Ustunlar soni>];

Ikkita satr va to'rtta ustunni olgan ikki o'lchovli massivni e'lon qilish:

int mas[2][4]; *// har biri 4 elementli ikita satr*

Ikki o'lchovli massivning barcha elementlari xotirada birbiridan keyin joylashadi. Avval birinchi satr elementlari, so'ngra ikkinchi va hakozo. Ikki o'lchovli massivni nomlashda elementlar qavs ichida vergul bilan ajratilib ko'rsatiladi. Misollar:

int mas[2][4]={
1, 2, 3, 4,
5, 6, 7, 8
};
	int mas[2][4]={
	{1, 2, 3, 4},
	{5, 6, 7, 8}
	};
		int mas[][4]={
		{1, 2, 3, 4},
		{5, 6, 7, 8}
		};

Nomlash jarayoni ko'rgazmali bo'lishi uchun elementlar alohida satrlarda keltirilgan. Satrdagi elementlar soni massivning ustunlari soni bilan mos keladi. Agar bir satr elementlari soni figurali qavslar ichida joylashtirilsa, yanada ko'rgazmali bo'ladi.

E'lon qilish davrida nomlash jarayoni ham amalga oshirilsa, satrlar sonini ko'rsatish shart emas, u avtomatik aniqlanadi.

Ikkita indeksni ko'rsatib, massiv elementining qiymatini berish yoki olish mumkin (nomerlash noldan boshlanadi). Yuqorida keltirilgan misolda elementlarni chop qilish quyidagicha ko'rinishda bo'ladi:

```
cout << mas[0][0] << endl; // 1
cout << mas[1][0] << endl; // 5
```

```
cout << mas[1][3] << endl; // 8
```

Massivning barcha qiymatlarini chiqarish uchun ichma-ich joylashgan sikllardan foydalanish zarur. Masalan, massivning barcha elementlarini nomerlash va barcha qiymatlarni chop qilish dasturini tuzamiz.

```
const short ROWS = 4; const short COLS = 5;
int i, j, n = 1, mas[ROWS][COLS];
 // Massivning barcha elementlarini nomerlash
for(i = 0; i < ROWS; i++) {
 for(j = 0; j < COLS; j++) {
 mas[i][j] = n;
 ++n;
 }
}
 // Qiymatlarni chiqarish
for(i = 0; i < ROWS; i++) {
 for(j = 0; j < COLS; j++) {
 cout.width(5); // Ma'lumot maydoni kengligi
 cout << mas[i][j];
 }
 cout << endl;
}
```

```
    1    2    3    4    5
    6    7    8    9   10
   11   12   13   14   15
   16   17   18   19   20
```

Ko‘p o‘lchamli massivning barcha elementlari xotirada bir-biridan keyin joylashadi. Shu sababli massivni saralashda ko‘rsatkichdan foydalanish mumkin.

Ikki o‘lchovli massiv elementlarini for sikli va ko‘rsatkich yordamida chiqarish dasturi listingi.

```
const short ROWS = 4, COLS = 5;
int mas[ROWS][COLS] = {{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10},
```

```
{11, 12, 13, 14, 15}, {16, 17, 18, 19, 20}};
for(int *p = mas[0]; p < mas[0] + ROWS * COLS; p++) {
cout << *p << endl;
}
```

6.4. Nomlar fazosi

Nomlar fazosi dastur ichida identifikatorlar to'qnashuvining (konfliktini) oldini olish uchun mo'ljallangan. Masalan, dasturingizda sum() nomli funksiya e'lon qilingan. Boshqa ishlab chiquvchining kutubxonasidan foydalanish zarurati yuzaga keldi. Bu kutubxonada ham sum() nomi bilan funksiya e'lon qilingan. Kutubxona ulanganida nomlar to'qnashuvi yuzaga keladi. Global sohada identifikatorlar qancha kam bo'lsa, dasturda to'qnashuvlar shucha kam bo'ladi. Bunday to'qnashuv standart kutubxonadan foydalanilganda ham yuzaga kelgan. Shu sababli, VC++ tilining standart kutubxonasi identifikatorlari std nomlar fazosiga joylashtirilgan. Bu identifikatorlarga kirish uchun uning nomidan oldin nomlar fazosini ko'rsatish zarur. Nomlar fazosi va identifikator o'rtasiga "::" operatori qo'yiladi: Masalan:

```
std::cout << "Konsol oynasiga kiritiluvchi matn";
```

Standart kutubxonaning "h" kengaytmasiga ega bo'lgan sarlavha fayllari barcha identifikatorlarni global nomlar fazosiga joylashtiradi. Nomlar fazosini e'lon qilish uchun namespace kalit so'zidan foydalaniladi. Nomlar fazosi global ko'rinish sohasida e'lon qilinishi lozim.

Nomlar fazosini e'lon qilish formati:

```
namespace <Nomlar fazosi nomi> {
// Identifikatorlarni e'lon qilish
}
```

NS nomlar fazosi va uning ichida "x" o'zgaruvchini e'lon qilishga misol ko'raylik:

```
namespace NS {
int x = 10;
```

```
}
```

Dasturda "x" o'zgaruvchiga kirish uchun o'zgaruvchidan oldin nomlar fazosining nomini ko'rsatish va "::" operatorini qo'yish lozim. Nomlar fazosini bir necha marta, hamda ichma-ich e'lon qilish mumkin, masalan:

```
cout << NS::x << endl;
namespace NS {
namespace NS2 {
int y = 30;
}
}
```

Ichki e'lon qilingan nomlar fazosidagi o'zgaruvchilarga kirish uchun identifikator nomidan oldin bir necha nomlar fazosi keltiriladi:

```
cout << NS::NS2::y << endl;
```

Global nomlar fazosida e'lon qilingan identifikatorlarga kirish uchun identifikatorlardan oldin faqat "::" operatori ko'rsatiladi. Masalan:

```
cout << ::z << endl;
```

using kalit so'zi yordamida barcha yoki ma'lum bir identifikatorlarni nomlar fazosidan global ko'rish sohasiga import qilish mumkin. Barcha identifikatorlarni import qilish quyidagicha:

```
using namespace <Nomlar fazosi nomi>;
```

Alohida identifikatorlarni import qilishda quyidaficha formatdan foydalaniladi:

```
using <Nomlar fazosi nomi>::<Identifikator>;
```

Nomlar fazosidan foydalanishga misol.

```
#include <iostream>
namespace NS {
 int x = 10;
 void print() {
```

```
/* 'x' shu ko'rinish sohasida joylashganligi sababli \
undan oldin nomlar fazosini ko'rsatish zarur emas.*/
cout << x << endl;
}
}
namespace NS { // Fazo ikki marta e'lon qilingan
 int y = 20;
namespace NS2 { // Ichki nomlar fazosi
int z = 30;
}
}
int main() {
 cout << NS::x << endl; // 10
 NS::print (); // 10
 // Ichki nomlar fazosi identifikatorlariga kirish
 cout << NS::NS2::z << endl; // 30
 // Faqat "y" inentifikatorini import qilish
 using NS::y;
 cout << "y = " << y << endl; // 20
 // Barcha inentifikatorlarni import qilish
 using namespace NS;
 cout << "x = " << x << endl; // 10
 print(); // 10
 cin.get();
 return 0;
}
```

```
10
10
30
y = 20
x = 10
10
```

VC++ tilida nomlar fazosini nomlamaslik ham e'tiborga olingan. Bunday nomlar fazosida e'lon qilingan identifikatorlar faqat fayl doirasida ko'rinadi. Nomlanmagan nomlar fazosini e'lon qilish formati quyidagicha:

```
namespace {
// Identifikatorlarni e'lon qilish
}
```

Nomlanmagan nomlar fazosi identifikatorlariga murojaat qilish global identifikatorlarga murojaat qilish kabi amalga oshiriladi. Masalan,

```
#include <iostream>
/* Nomlanmagan nomlar fazosining ko'rinish sohasi
fayl bilan chegaralangan */
namespace {
 int x = 10;
}
int main() {
 cout << x << endl; // 10
 cin.get();
 return 0;
}
```

7. Funksiyalar

Funksiya – dasturning ihtiyoriy joyidan bir nech bor chaqirish mumkin bo'lgan kod qismi. Yuqorida ko'rib o'tilgan mavzularda standart kutubxona tarkibiga kiruvchi biriktirilgan funksiyalardan bir necha bor foydalanildi. Masalan, strlen() funksiyasi yordamida C-satrdagi belgilar soni olindi. Bu bo'limda dastgur kodining ortiqchaligini kamaytirish imkonini beruvchi va tarkiblanganlik darajasini orttirish uchun xizmat qiluvchi foydalanuvchi funksiyalarini ko'rib o'tamiz.

7.1. Funikiyani yaratish va chaqirish

Funksiyaning bayoni ikki qismdan iborat: e'lon va aniqlanish. Funksiyani e'lon qilish (*funksiyaning prototipi deb ham ataladi*) tip haqidagi ma'lumotni oladi. Axborotni olgan kompilyator tipning va parametrlar sonining mos kelmasligini aniqlashi mumkin. Funksiya prototipi formati quyidagicha:

<Natija tipi> <Funksiya nomi>([<Tip> [<Parametr nomi1>]
[, ..., <Tip> [<Parametr nomiN>]]]);

<Natija tipi> parametri funksiya return operatori yordamida qaytaradigan qiymat tipini beradi. Agar funksiya qiymat qaytarmasa, tip o'rniga void kalit so'zi ko'rsatiladi.

Funksiya nomi, o'zgaruvchilar nomlariga qo'yiladigan talablar kabi, mumkin bo'lgan identifikator bo'lishi kerak. Funksiya nomidan keyin qavs ichida tipi va vergullar bilan ajratilgan holda parametrlar nomlari ko'rsatiladi. Funksiya prototipidagi parametrlarni umuman ko'rsatmaslik ham mumkin, chunki kompilyatorni faqat berilganlar tipi va parametrlar soni qiziqtiradi. Funksiya parametrga ega bo'lmasa, faqat qavslar yoki qavs ichida void kalit so'zi ko'rsatiladi (C tilida void kalit so'zi majburiy hisoblanadi). Funksiya e'lonidan keyin nuqtali vergul turishi kerak. Funksiyani e'lon qilishga misol:

int sum(int x, int y); *// yoki int sum(int, int);*

void print(const char *str); *// yoki void print(const char *);*

void print_ok(); *// yoki void print_ok(void);*

Funksiya aniqlanishi tip bayoni va parametr nomi, hamda amalga oshirilishini oladi. Funksiyaning aniqlanishi quyidagicha formatga ega:

<Natija tipi> <Funksiya nomi>([<Tip> [<Parametr nomi1>]
[, ..., <Tip> [<Parametr nomiN>]]) {
<Funksiya tanasi>
[return[<Qaytaruvchi qiymat>];]
}

Funksiyaning aniqlanishida, prototipdan farqli ravisha, lokal o'zgaruvchi hisoblanuvchi parametrning nomini ko'rsatish majburiy. Bu o'zgaruvchi funksiya chaqirilganda yaratiladi, funksiyadan chiqilgandan keyin o'chiriladi. Lokal o'zgaruvchining nomi global o'zgaruvchi bilan mos kelsa, u hoda barcha amallar lokal o'zgaruvchi bilan amalga oshiriladi, global o'zgaruvchining

qiymati o'zgarmaydi. Bunda global o'zgaruvchiga murojaat qilish uchun nomdan oldin "::" operatorini ko'rsatish zarur.

```
int sum(int x, int y) {
 int z = x + y; // "x" lokal o'zgaruvchisiga murojaat
 z = ::x + y; // "x" global o'zgaruvchisiga murojaat
 return z;
}
```

Parametrlar bayonidan sog, figurali qavs ichida funksiyaning har bir chaqirilishida bajariluvchi ko'satmalar joylashadi. har qanday holatda ham (tanada bitta ko'rsatmadan iborat bo'lsa ham) figurali qavslar ko'rsatiladi. Yopiluvchi qavsdan keyin nuqtali vergul qo'yilmaydi.

Funksiyadan qiymat qaytarish uchun return operatoridan foydalaniladi. Bu operator bajarilganidan so'ng, funksiyaning bajarilishi to'xtaydi va boshqaruv funksiyani chaqirish nuqtasiga uzatiladi. Demak, return opeatoridan keyin joylashgan ko'rsatmalar bajarilmaydi. return operatoridan foydalanishda turlicha vaziyatlar bo'lmasligi lozim. Masalan, quyidagi holatda qaytariluvchi qiymat qo'yilgan shartga bog'liq.

```
int sum(int x, int y) {
 if(x > 0) {
 return x + y;
 }
}
```

Agar 'x' o'zgaruvchi noldan katta qiymatga ega bo'lsa, hammasi joyida, aks holda qaytariluvchi qiymat aniqlanmagan va funksiya tasodifiy qiymat ("musor" deb nomlanuvchi) qaytaradi. Bunday holda kompilyatsiya jarayonida ogohlantiruvchi xabar chiqadi: "*warning C4715 sum* значение возвращается не при всех путях выполнения". Bunday turlilikni chetlab o'tish uchun funksiya tanasi oxirida ko'rsatilmagandagi qiymat bilan return operatorini qo'yish tavsiya qilinadi:

```
int sum(int x, int y) {
```

```
if(x > 0) {
return x + y;
}
return 0;
}
```

Funksiya nomidan oldin void kalit so'zi ko'rsatilgan bo'lsa, return operatori bo'lmasligi mumkin. Ammo, funksiyaning bajarilishini muddatidan oldin tugallash zarur bo'lsa, u holda return operatori qaytariluvchi qiymatsiz ko'rsatiladi. Masalan,

```
void print_ok() {
cout << "OK" << endl;
return; // Funksiyaning bajarilishini muddatidan oldin tugallash
cout << "Bu ko'rsatma hech qachon bajarilmaydi"
}
```

Funksiya dasturdan chaqirilganda uning nomi ko'rsatiladi, so'mgra qavs ichida qiymatlar uzatiladi. Funksiyac parametrlar qabul qilmasa, faqat qavslar ko'rsatiladi. Agar funksiyac qiymat qaytaradigan bo'lsa, uni o'zgaruvchiga o'zlashtirish yoki shunchaki e'tidorga olmaslik mumkin. Parametrlarning tipi va soni chaqiruvdagi parametrlar va tip bilan mos kelishi lozim.

Funksiyalarni chaqirishga misollar ko'raylik:

```
print("Message"); // Funksiya xabar chiqaradi
Message print_ok(); // Parametrsiz funksiyani chaqirish
z = sum(10, 20); // "z" ga '30' qiymat o'zlashtiriladi
```

Uzatilgan qiymatlar mos ravishda funksiya aniqlanishida joylashgan o'zgaruvchilarga o'zlashtiriladi. sum() funksiyasidan foydalanilganda x o'zgaruvchiga 10 qiymati, y o'zgaruvchiga esa 20 qiymati o'zlashtiriladi. Funksiyaning bajarilishi z o'zgaruvchiga o'zlashtiriladi.

Misol tariqasida uchta turli funksiya yaratamiz va ularni chaqiramiz.

```
#include <iostream>
```

```
// Funksiyalarni e'lon qilish
int sum(int x, int y); // yoki int sum(int, int);
void print(const char *str); // yoki void print(const char *);
void print_ok(); // yoki void print_ok(void);
int main() { // Funksiyalarni chaqirish
 print("Message"); // Message
 print_ok(); // OK
 cout << sum(10, 20) << endl; // 30
 cin.get();
 return 0;
}
// Funksiyalarning aniqlanishi
int sum(int x, int y) { // Ikkita parametr
 return x + y;
}
void print(const char *str) { // Bitta parametr
 cout << str << endl;
}
void print_ok() { // Parametrsiz
 cout << "OK" << endl;
}
```

7.2. Funksiyaning e'loni va aniqlanishi

Dasturdagi barcha ko'rsatmalar yuqoridan pastga qarab bajariladi. Bu shuni anglatadiki, dasturda funksiyadan foydalanishdan avval uni qo'shimcha ravishda e'lon qilish kerak. Shu sababli funksiyaning e'loni funksiyani chaqirishdan oldin joylashishi lozim. Bir funksiyaning aniqlanishini ikkinchi funksiyaning ichida joylashtirish mumkin emas. Funksiyaning nomi har doim global identifikator hisoblanadi.

Katta bo'lmagan dasturlarda funksiya e'lonini ko'rsatmasdan main() funksiyasidan oldin funksiyaning aniqlanishini joylashtirish mumkin. main() funksiyasi ham, birinchi chaqilirgani uchun, e'lonni talab qilmaydi.

```
#include <iostream>
int sum(int x, int y) {
 return x + y;
}
int main() {
 cout << sum(10, 20) << endl; // 30
 cin.get();
 return 0;
}
```

Funksiyalarning e'lonini main() funksiyasidan oldin, bayonini esa funksiyadan keyin joylashtirish lozim. Bunda funksiyalarning aniqlanish tartibi ahamiyatga ega emas.

```
#include <iostream>
int sum(int, int); // Funksiya e'loni
int main() {
 cout << sum(10, 20) << endl; // 30
 cin.get();
 return 0;
}
int sum(int x, int y) { // Aniqlanishi
 return x + y;
}
```

Dastur o'lchami ortishi bilan e'lonlar va aniqlanishlar ortib boradi. Bunday holda dastur bir necha alohida fayllarga bo'linadi. Funksiya e'lonlari h (ayrim hollarda hpp) kengaytmali sarlavha fayliga, aniqlanishlar esa bir nomli cpp kengaytmali alohida faylga chiqariladi. Barcha fayllar main() funksiyasini olgan asosiy fayl bilan birga bitta papkada joylashtiriladi. #include direktivasi yordamida sarlavha fayli barcha fayllarga ulanadi. Agar #include direktivasida sarlavha fayli nomi burchakli qavslar ichida ko'rsatilsa, fayl tizim papkalaridan qidiriladi, qo'shtirnoq ichida ko'rsatilsa, avval asosiy fayl joylashgan papkadan, so'ngra tizim papkalaridan qidiradi.

Misol sifatida sum() funksiyasini alohida faylga (masalan, "mymodule" nomi bilan) chiqaramiz, so'ngra uni asosiy faylga ulaymiz. mymodule.cpp faylini yaratish uchun **Обозреватель решений** oynasida sichqoncha o'ng tugmasi bosilib, **Файлы исходного кода** punkti va kontekstli menyudan **Добавить=>Создать элемент** punkti tanlanadi. Natijada **Добавление нового элемента** oynasi ochiladi. **Файл** *C++ (.cpp)* punkti tanlanib, fayl nomi ("mymodule") kiritiladi va **Добавить** tugmasi bosiladi. Fayl loyiha papkasiga qo'shiladi va uning nomi **Обозреватель решений** oynasida tasvirlanadi. Faylning o'zi esa alohida qo'yilmada ochiladi. Bu faylga quyidagi kodni qo'yamiz (mymodule.cpp fayli):

```
#include "mymodule.h"
int sum(int x, int y) { // Aniqlash
return x + y;
}
```

mymodule.h faylini yaratish uchun **Обозреватель решений** oynasining **Заголовочные файлы** punktida sichqoncha o'ng tugmasi bosiladi va kontekstli menyudan **Добавить=>Создать элемент** punkti tanlanadi. Natijada **Добавление нового элемента** oynasi ochiladi. Oynada **Заголовочный файл** (.h, fayl nomi kiritiladi - "mymodule") punkti tanlanib, **Добавить** tugmasi bosiladi. Bu faylga quyidagi kod joylashtiriladi (mymodule.h fayli):

```
#ifndef MYMODULE_H
#define MYMODULE_H
#include <iostream>
int sum(int x, int y) ;
#endif
```

mymodule.h faylining mazmuni kompilyatsiyadan oldin tekshiriladigan shart ichida joylashgan. Shart quyidagicha ko'rinishga ega:

```
#ifndef MYMODULE_H
```

```
// Ko'rsatmalar
#endif
```

Shartni quyidagicha o'qish mumkin: "agar MYMODULE_H makroaniqlanishi mavjud bo'lmasa, u holda ko'rsatmalar fayl ulanayotgan joyga qo'yilsin". Odatda makroaniqlanish nomi sarlavha fayli nomi bilan mos keladi. Faqat barcha harflar yuqori registrda ko'rsatiladi va nuqta ostki chiziq bilan almashtiriladi. Shart #ifndef direktivasidan boshlanadi va #endif direktivasi bilan tugaydi. Bu identifikatorlar e'loni ikkki marta qo'yilmasligi uchun zarur. Buning uchun birinchi ko'rsatmada shart ichida #define direktivasi yordamida MYMODULE_H makroaniqlanish e'lon qilinadi. Bu holda takroriy tekshirish yolg'on qiymat qaytaradi. Faylni bir marta ulash uchun direktivasidan ham foydalanish mumkin #pragma: #pragma once

Bu sum() funksiyasidan foydalanish uchun asosiy dasturga mymodule.h faylini ulash yetarli bo'ladi. Ulash quyidagi listingda (main.cpp) keltirilgan:

```
#include "mymodule.h"
int main() {
 cout << sum(10, 20) << endl; // 30
 cin.get() ;
 return 0;
}
```

#include direktivasida nafaqat faylning nomini, balki unga absolyut yoki nisbiy yo'lni ko'rsatish ham mumkin. Bu fayllarni turli papkalarda joylashtirish imkonini beradi. Absolyut va nisbiy yo'lni korsatish:

```
#include "C:\\book\\test\\test\\mymodule.h"
#include "C:/book/test/test/mymodule.h"
#include "/book/test/test/mymodule.h"
#include "folder/mymodule.h"
```

Katta dasturlardan foydalanilganda statik (VC++da lib kengaytmali fayllar) yoki dinamik (dll kengaytmali fayllar) kutubxonalar yaratiladi. Statik kutubxonalar dastur yoki kom-

pilyatsiyaning bir qismiga aylanadi, dinamik kutubxonalar esa dasturning bajarilishi vaqtida yuklanadi.

7.3. Biriktirilgan funksiyalar

Boshqaruvni funksiyaga berish dastur bajarilishi tezligining pasayishi bilan kechadi. Chunki funksiyani chaqirish, unga parametrlarni uzatish va qiymatni qaytarish qo'shimcha vaqtni talab qiladi. Agar funksiyaning o'lchami katta bo'lmasa, bunday funksiyani *biriktirilgan* holda e'lon qilish maqsadga muvofiq. Bunda kompilyatsiya jarayonida funksiyaning mazmuni funksiyani chaqirish o'rniga qo'yiladi. Bunda bajariluvchi fayl o'lchamining hajmi ortadi. Shu sababli, katta funksiyani ko'p marotaba chaqirish lozim bo'lsa, uni oddiy funksiya ko'rinishida qoldirgan ma'qul. Bi ko'rsatmadan tashkil topgan funksiya biriktirish uchun birinchi nomzod hisoblanadi.

Funksiyani biriktirilgan holda e'lon qilish uchun funksiyadan oldin inline kalit so'zini qo'shish lozim. inline kalit so'zi kompilyator uchun tavsiya hisoblanadi va e'tiborga olinmasligi mumkin. Misol sifatida sum() biriktirilgan funksiyasini e'lon qilamiz:

```
Biriktiriluvchi funksiyalar
#include <iostream>
inline int sum(int x, int y);
int main() {
 cout << sum(10, 20) << endl;
 cin.get();
 return 0;
}
inline int sum(int x, int y) {
 return x + y;
}
```

Biriktiriluvchi funksiyani #define direktivasi yordamida ham yaratish mumkin. Bu direktivada ko'rsatilgan qiymat kompil-

yatsiyaga qadar funksiyani chaqiruv o'rniga qo'yiladi. #define direktivasida ko'rsatilgan nomni *makroaniqlanish* yoki *makros* deb nomlash qabul qilingan.

Direktivaning formati quyidagicha:

#define <Funksiya nomi>(<Parametrlar>) <Ko'rsatma>

```
// #define direktivasidan foydalanish
#include <iostream>
#define SUM(x, y) (x + y)
int main() {
 cout << SUM(10, 20) << endl;
 cin.get();
 return 0;
}
```

Ko'rsatma oxirida nuqtali vergul ko'rsatilmaydi. Ko'rsatmaning oxiri satrning oxiri hisoblanadi. Agar nuqtali vergul qo'yilsa, qiymat u bilan birga ifodaga qo'yiladi. Masalan, agar makros

define SUM(x, y) (x + y);

ko'rinishda aniqlansa, u holda qiymat qo'yilganidan keyin

cout << SUM(10, 20) << endl;

ko'rsatma quyidagicha ko'rinishga ega bo'ladi:

cout << (10 + 20); << endl;

Yopiluvchi qavsdan keyingi nuqtali vergul ko'rsatmaning oxiri hisoblanadi. Shu sababli, kompilyatsiyada bu kod xatolikni chaqiradi.

int z = SUM(10, 20) * 40;

ko'rinishda yoziladigan bo'lsa, xatolik yuzaga kelmaydi. Qiymatlar qo'yilganida quyidagicha ko'rinishga ega bo'ladi:

int z = (10 + 20); * 40;

Bunday vaziyat topish qiyin bo'lgan xatolikka olib keladi, chunki bunday holda kompilyatsiya jarayonida " * 40;" ko'rsatmasi xatolikni chaqirmaydi.

Makros tanasidagi barcha ifodalar formal parametrlar qo'yilganidan so'ng, butunligicha makrosni chaqirish joyiga qo'yiladi. Masalan, makros quyidagicha aniqlangan bo'lsin:

#define SUM(x, y) x + y

U holda qiymat qo'yilganidan so'ng

int z = SUM(10, 20) * 40;

ko'rsatmasi quyidagicha ko'rinishda bo'ladi:

int z = 10 + 20 * 40;

Natija 1200 soni o'rniga 810 soniga teng bo'ladi.

Funksiya tanasi ichida uzun ifodani ko'rsatishda makrosning aniqlanishi bir satrda joylashishini e'tiborga olish lozim. agar ifodani bir nechta satrda joylashtirish zarur bo'lsa, satr oxirida teskari og'ma chiziq qo'yish zarur. og'ma chiziqdan keyin hech qanday belgi bo'lmasligi kerak, hattoki izoh ham.

7.4. Parametrlar bilan ishlash

Parametrlarni funksiyaga berish usullari. Biz bilamizki, funksiya nomidan keyin qavs ichida tip va vergullar bilan ajratilgan parametrlar nomi ko'rsatiladi. Agar funksiya parametrlar qabul qilmasa, faqat qavslar yoki qavs ichida void kalit so'zi ko'rsatiladi (C tilida void so'zi majburiy hisoblanadi). Funksiyaning aniqlanishidatipdan tipdan keyin lokal o'zgaruvchi hisoblangan parametrning nomi ko'rsatilishi zarur, bu o'zgaruvchi funksiyani chaqirishda yaratiladi va funksiyadan chiqilgandan keyin o'chiriladi.

Lokal o'zgaruvchining nomi global o'zgaruvchi nomi bilan mos kelsa, u holda barcha amallar lokal o'zgaruvchi bilan amalga oshiriladi, global o'zgaruvchining qiymati o'zgarmaydi. Bunda global o'zgaruvchiga murojaat qilish uchun uning nomidan oldin "::" operatori ko'rsatiladi.

Funksiyani chaqirishda uning nomi ko'rsatiladi, undan keyin qavs ichida qiymatlar uzatiladi. Funksiyaning aniqlanishida parametrlar soni va tipi chaqirishdagi soni va tipi bilan

mos kelishi kerak, aks holda xatolik yuzaga keladi. Uzatilgan qiymatlar mos o'rinlardagi o'zgaruvchilarga o'zlashtirilaid. Yuqorida keltirib o'tilgan misolda sum (10, 20) (int sum (int x, int y) prototipi) funksiyasini chaqirishda x o'zgaruvchiga 10, y o'zgaruvchisiga esa 20 qiymati o'zlashtiriladi.

Ko'rsatilmaganda funksiyaga ko'rsatkichga murojaat emas, balki o'zgaruvchi qiymatining nusxasi uzatiladi. Funksiya ichida qiymatning o'zgarishi boshlang'ich o'zgaruvchining qiymatiga ta'sir qilmaydi. Quyida parametrni qiymati bo'yicha uzatish listinggi keltirilgan.

```
#include <iostream>
void func(int x);
int main() {
 int x = 10;
 func(x);
 cout << x << endl; // 10, 30 emas
 cin.get()
 return 0;
}
void func(int x) {
 x = x + 20; // Qiymat hech qayerda saqlanmaydi!
}
```

Sonlar uchun qiymatning nusxasini uzatish yaxshi yechim hisoblanadi. Massiv va obyektlardan foydalanilganda, hamda boshlang'ich o'zgaruvchining qiymatini o'zgartirish zarurati yuzaga kelganda ko'rsatkichni uzatishni qo'llagan ma'qul. Buning uchun funksiyani chaqirishda o'zgaruvchidan oldin & (manzilni olish) operatori ko'rsatiladi, funksiya prototipida esa ko'rsatkich e'lon qilinadi. Bunday e'londa funksiyaga o'zgaruvchining qiymati emas, balki manzili (adresi) uzatiladi. Funksiya ichida o'zgaruvchi o'rniga ko'rsatkichdan foydalaniladi. Misol,

```
#include <iostream>
```

```
void func(int *y);
int main() {
 int x = 10;
 func(&x); // Manzil uzatiladi
 cout << x << endl; // 30, 10 emas
 cin.get();
 return 0;
}
void func(int *y) {
 *y = *y + 20; // "x" o'zgaruvchining qiymati o'zgaradi
}
```

Agar funksiya katta bo'lsa, u holda har bir amalda ko'rsatkishni qayta nomlash qulay emas. C++ tilida parametrlarni uzatishning yana bir mexanizmi mavjud – *murojaat mexanizmi*. Buning ma'nosi shundaki, funksiyaning ichida o'zgaruvch boshlang'ich o'zgaruvchining soxtasi hisoblanadi. Soxtaning har qanday o'zgarishi boshlang'ich o'zgaruvchida akslanadi. Murojaatlar mexanizmidan foydalanilganda e'londa va funksiyaning aniqlanishida o'zgaruvchi nomidan oldin & operatori ko'rsatiladi, chaqirishda esa – faqat o'zgaruvchining nomi.

Murojaatlar mexanizmidan foydalanishga misol ko'ramiz.

```
#include <iostream>
void func(int &y);
int main() {
 int x = 10;
 func(x);
 cout << x << endl; // 30, 10 emas
 cin.get();
 return 0;
}
void func(int &y) {
 // 'y' o'zgaruvchi 'x' o'zgaruvchining soxtasi hisoblanadi
 y = y + 20; // "x" o'zgaruvchining qiymati o'zgaradi
}
```

Majburiy bo'lmagan parametrlar. Ba'zi bir parametrlarni majburiy qilmaslik uchun funksiya e'lonida unga boshlang'ich qiymat o'zlashtiriladi. Bunday holda funksiya chaqirilganda parametr ko'rsatilmasa, o'zgaruvchiga boshlang'ich qiymat o'zlashtiriladi. Boshlang'ich qiymat faqat funksiya e'lonida beriladi. funksiyaning aniqlanishida uni takrorlamaslik lozim. Bundan tashqari, majburiy bo'lmagan parametrlar majburiy parametrlardan keyin kelishi lozim, aks holda xatolik haqida xabar chiqariladi. Misol sifatida sum() funksiyasining ikkinchi parametrini majburiy bo'lmagan qilamiz:

```
#include <iostream>
int sum(int x, int y = 2); // Boshlang'ich qiymat
int main() {
 cout << sum(10, 20) << endl; // 30
 cout << sum(10) << endl; // 12
 cin.get();
 return 0;
}
int sum(int x, int y) { // Aniqlashda boshlang'ich qiymat ko'rsatilmaydi
return x + y;
}
```

O'zgarmas parametrlar. Agar funksiya ichida parametr qiymati o'zgarmasa, bunday parametrni o'zgarmas sifatida e'lon qilish lozim. Buning uchun e'lon qilishda parametr oldida const so'zi ko'rsatiladi. Masalan, sonlarni yig'ish uchun mo'ljallangan sum() funksiyasi parametrlar qiymatining o'zgarishini amalga oshirmasa, parametrlarni o'zgarmas sifatida e'lon qilish mumkin:

```
int sum(const int x, const int y) {
 return x + y;
}
```

Ko'rsatkichlardan foydalanilganda const kalit so'zining joylashuvini hisobga olish muhim. Masalan, quyidagi e'lonlar ekvivalent emas:

```
void print(const char *s);
void print(char const *s);
void print(char *const s);
void print(const char *const s);
```

Birinchi ikkita e'lon ekvivalent hisoblanadi. Bunday holda ko'rsatkich murojaat qilayotgan qiymatni o'zgartirish mumkin emas, ammo ko'rsatkichga boshqa manzil o'zlashtirish mumkin.

```
void print(const char *s) {
 char s2[] = "New";
 cout << s << endl;
}
```

Uchinchi e'londa ko'rsatkich murojaat qilayotgan qiymatni o'zgartirish mumkin, ammo ko'rsatkichga boshqa manzilni o'zlashtirish mumkin emas:

```
void print(char * const s) {
 char s2[] = "New";
 s = s2; // Xato
 s[0] = 's'; // Normal holat
 cout << s << endl;
}
```

To'rtinchi e'lon ko'rsatkich murojaat qilayotgan qiymatni o'zgartirishni va boshqa manzil o'zlashtirishni ta'qiqlaydi:

```
void print(const char *const s) {
 char s2[] = "New";
 s = s2; // Xato
 s[0] = 's'; // Xato
 cout << s << endl;
}
```

const_cast operatori orqali const va volatile kalit so'zlarining ta'sirini (ishlashini) bekor qilish mumkin. Operatorning formati:

const_cast< <Natija tipi> > (<Ifoda>)

Funksiya ichida o'zgarmas ko'rsatkichni oddiyga keltirishga doir misol keltiramiz:

```
void print(const char *s) {
 //s[0] = 's'; // Xato
 char *p = const_cast<char *> (s);
 p[0] = 's'; // Normal holat
 cout << s << endl;
}
```

Funksiya ichidagi murojaatda const kalit so'zining ishlashini o'chirish (yo'qotish) ga misol:

```
void func(const int &x) {
 //x = 50; // Xato!
 // Endi qiymatni o'zgartirish mumkin
 const_cast<int &>(x) = 30;
}
```

O'zgaruvchan sondagi parametrlar. Funksiyada parametrlar soni, bitta majburiy parametr mavjudlik shartida, ihtiyoriy sonda bo'lishi mumkin. Funksiyaning e'loni va aniqlanishida o'zgaruvchan sondagi parametrlar uchta nuqta orqali belgilanadi. Funksiya ichida bu parametrlarga kirish uchun bir necha usulidan foydalanish mumkin.

Birinchi usul parametrlar sonini majburiy parametrda uzatish. Bunda oxirgi majburiy parametr manzili ko'rsatkichda saqlanadi. Ko'rsatkich yordamida keyingi parametrga o'tish bajariladi. Misol tariqasida ihtiyoriy sondagi butun sonlar yig'indisini topish dasturini yozamiz.

```
#include <iostream>
int sum(int x, ...);
int main() {
 cout << sum(2, 20, 30) << endl; // 50
 cout << sum[3, 1, 2, 3) << endl; // 6
 cout << sum(4, 1, 2, 3, 4) << endl; // 10
```

```
cin.get();
return 0;
}
int sum(int x, ...) {
 int result = 0;
 int *p = &x; // Oxirgi parametr manzilini olamiz
 for(int i = 0; 1 < x; i++) {
 ++p; // Ko'rsatkichni navbatdagi parametrga ko'chiramiz
 result += *p; // Navbatdagi sonni qo'shamizs
 }
return result;
}
```

Bu misolda majburiy parametr tipi boshqa parametrlar tipi bilan mos keladi. Ihtiyoriy sondagi haqiqiy sonlar yig'indisini topish talab qilinsa, uning yordamida parametrlar soni uzatilayotgan majburiy parametr tipini o'zgartirish yoki ko'rsatkichning ko'chishida tiplarni keltirishni bajarish mumkin.

Ihtiyoriy sondagi haqiqiy sonlarni qo'shish.

```
#include <iostream>
double sum(int x, ...);
int main() {
 cout << sum(2, 20.2, 3.6) << endl; // 23.8
 cout << sum(3, 1., 2.8, 3.4) << endl; // 7.2
 cin.get();
 return 0;
}
double sum(int x, ...) {
 double result = 0.0, *pd = 0;
 int *pi = &x; // Oxirgi parametr manzilini olamiz
 ++pi; // Ko'rsatkichni tiplarni keltirishdan avval
ko'chiramiz !!!
 pd = reinterpret_cast<double *> (pi); // Tiplarni keltirish
 for(int i = 0; i < x; i++) {
```

```
    result += *pd; // Navbatdagi sonni qo'shamiz
    ++pd; // Ko'rsatkichni navbatdagi parametrga ko'chiramiz
    }
    return result;
    }
```

Funksiyaga ihtiyoriy sondagi satrni uzatishda oxirgi parametrda nol ko'rsatkichni berish mumkin. Bunday holda paraametrlar sonini oshkor berish shart emas. Misol tariqasida funksiyaga bir necha satr uzatamiz. Konsol oynasida satrlar mazmunini funksiya ichida chiqaramiz.

```
    #include <iostream>
    void func(char *s, ...);
    int main() {
     char str1[] = "string1", str2[] = "string2";
     func(str1, str2, 0); // Oxirgi parametrda nol ko'rsatkich
     func(str1, str2, str1, 0); // Oxirgi parametrda nol
ko'rsatkich
     cin.get();
     return 0;
    }
    void func(char *s, ...) {
     char **p = &s;
     while (*p) {
     cout << *p << endl;
     ++p;
     }
    }
```

```
string1
string2
string1
string2
string1
```

7.5. Massivlarni funksiyaga uzatish

Bir o'lchovli massivlar va C-satrlarni funksiyaga uzatish ko'rsatkichlar yordamida malga oshiriladi. Funksiyani chiqarishda o'zgaruvchi nomidan oldin '&' operatori qo'yilmaydi, chunki o'zgaruvchining nomi massiv birinchi elementining manzilini oladi. Quyidagi dastur kodida C-satrni funksiyaga uzatishga keltirilgan.

```
#include <iostream>
void func1(char *s);
void func2(char s[]);
int main() {
 char str[] = "String";
 cout << sizeof(str) << endl; // 7
 func1(str); // str dan oldin & operatori ko'rsatilmaydi
 func2(str);
 cout << str << endl; // string
 cin.get();
 return 0;
}
void func1(char *s) {
 s[0] = 's'; // str massiv elementi qiymati o'zgaradi
 cout << sizeof(s) << endl; // 4, 7 emas!
}
void func2(char s[]) {
 s[5] = 'G'; // str massiv elementining qiymati o'zgaradi
}
```

```
7
4
stringG
```

char *s e'loni char s[] e'loniga ekvivalent. har ikki holatda ham char tipidagi ko'rsatkich e'lon qilinadi. sizeof operatori funksiyadan tashqarida belgili massivning butun o'lchamini, funksiya ichida faqat ko'rsatkichning o'lchamini qaytaradi. Chunki funksiya ichida s o'zgaruvchi massiv emas, ko'rsatkich hisoblanadi.

Ko‘p o‘lchovli massivni uzatishda barcha o‘lchamlari aniq ko‘rsatilishi lozim (masalan, int a[4][4][4]). Birinchi o‘lchamni ko‘rsatmaslik mumkin (masalan, int a[][4][4]).

Ikki o‘lchovli massivni funksiyaga uzatishga misol:

```
#include <iostream>
void func(int a[][4]);
int main() {
 const short ROWS = 2, COLS = 4;
 int i, j;
 int mas[ROWS][COLS] = {{1, 2, 3, 4}, {5, 6, 7, 8}};
 func(mas);
 // Qiymatlarni chiqaramiz
 for(i = 0; i < ROWS; i++) {
 for(j = 0; j < COLS; j++) {
 cout.width(4); // Maydonning kengligi
 cout << mas[i][j];
 }
 cout << endl;
 }
 cin.get();
 return 0;
}
void func(int a[][4]) { // yoki void func(int a[2][4])
 a[0][0] = 80;
}
```

```
80   2   3   4
 5   6   7   8
```

Ko‘p o‘lchovli massivni bunday uzarishni universal deb bo‘lmaydi, chunki qat’iy bog‘lanish mavjud. Buni hal qilishning yechlaridan biri qo‘shimcha ko‘rsatkichlar massivini yaratish hisoblanadi. Bunday hola funksiyaga ko‘rsatkichlar massivining birinchi elementi manzili beriladi, funksiyada parametrni e’lon qilish quyidagicha ko‘rinishda bo‘ladi:

int *a[] yoki int **a

Ko'rsatkichlar massivini funksiyaga uzatishga doir misol:

```
#include <iostream>
void func(int *a[], short rows, short cols);
int main() {
 const short ROWS=2, COLS=4;
 int mas[ROWS][COLS] = {{1, 2, 3, 4}, {5, 6, 7, 8}};
 // Ko'rsatkichlar massivini yaratish
 int *pmas[] = {mas[0], mas[1]};
 // Ko'rsatkichlar massivi manzilini uzatish
 func(par R, ROWS, COLS);
 cin.get();
 return 0;
}
void func(int *a[], short rows, short cols) {
 // void func(int **a, short rows, short cols)
 int i,j;
 for(i = 0; i < rows; i++) {
 for(j = 0; j < cols; j++) {
 cout.width(4);
 cout << a[i][j];
 }
 cout << endl;
 }
}
```

Bu usul ham kamchilikga ega, chunki qo'shimch ko'rsatkichlar massivini yaratish zarur. Eng yaxshi usul ko'p o'lchovli massivni bir o'lchovli kabi uzatish hisoblanadi. Bu holda funksiyaga massiv birinchi elementining manzili uzatiladi, parametrida ko'rsatkich e'lon qilinadi. Ko'p o'lchovli massivning barcha elementlari xotirada ketma-ket joylashganligi sababli, elementlar soni yoki o'lchamlarini bilgan holda joriy elementning joylashuvini manzilli arifmetikadan foydalanib hisoblash mumkin.

Ikki o'lchovli massivni bir o'lchovli sifatida uzatishga misol.

```
// Ikki o'lchovli massivni bir o'lchovli sifatida uzatish
#include <iostream>
void func(int *a, short rows, short cols);
int main() {
 const short ROWS = 2, COLS = 4;
 int mas[ROWS][COLS] = {{1 ,2, 3, 4}, {5, 6, 7, 8}};
 // Funksiyaga massiv birinchi elementining manzilini uzata-
miz
 func(mas[0], ROWS, COLS);
 cin.get();
 return 0;
}
void func(int *a, short rows, short cols) {
 int i, j;
 for(i = 0; i < rows; i++) {
 for(j = 0; j < cols; j++) {
 cout.width(4);
 cout << a[i * cols + j]; // element o'rnini hisoblash
 }
 cout << endl;
 }
}
```

C-satrlar massivini funksiyaga uzatish ko'rsatkichlar massivini uzatish kabi amalga oshiriladi. Funksiyaga elementlar sonini uzatmaslik uchun C-satrlar massivini nomlashda oxirgi elementiga nol ko'rsatkich berish mumkin. Bu element massiv oxirini ko'rsatuvchi belgi bo'lib xizmat qiladi. Misol tariqasida barcha satrlarni funksiya ichida chiqaramiz.

```
#include <iostream>
void func(char *s[]);
int main() {
 char *str[]={"String1", "String2", "String3", 0};
```

```
// Nolinchi ko'rsatkich qo'yamiz – oriyentir (oxirini
ko'rsatish) uchun
 func(str);
 cin.get();
 return 0;
}
void func(char *s[]) { // yoki void func(char **s)
 while(*s) { // Barcha satrlarni chiqarish
 cout << *s << endl;
 ++s;
 }
}
```

7.6. Funksiyani qayta yuklash

Shunday vaziyatlar yuzaga keladiki, haqiqiy sonlarni yig'ishni hisoblashga to'g'ri keladi. VC++ tilida bunday vaziyatni yechish uchun, funksiyani qayta yuklash deb nomlanuvchi, yechim mavjud. Funiksiyani qayta yuklash – bu bir nomdan, parametrlar tiplari yoki ularning soni bilan farqlanuvchi, bir necha funksiyalar uchun foydalanish imkoniyati. Faqat qaytariluvchi qiymat tipining o'zgarishi funksiyani qayta yuklash uchun yetarli emas. Misol tariqasida sum() funksiyasini shunday qayta yuklaymizki, undan ham butun, ham haqiqy sonlarni qo'shishda foydalanish mumkin bo'lsin.

```
#include <iostream>
int sum(int x, int y);
double sum(double x,double y) ;
int main() {
 // Butun sonlarni qo'shish
 cout << sum(10, 20) << endl; // 30
 // Haqiqiy sonlarni qo'shish
 cout << sum(10.5, 20.7) << endl; // 31.2
 cin.get();
 return 0;
```

```
}
int sum(int x, int y) {
 return x + y;
}
double sum(double x, double y) {
 return x + y;
}
```

Qayta yuklangan funksiyani chaqirishda turlilik (неоднозначность) yuzaga kelishi mumkin (kompilyator qaysi funksiyani chaqirishni tanlay olmaydi). Bunday vaziyatlarning sababchisi, qiymatni funksiyaga uzatishda tipni avtomatik almashtiruvchi, ko'rsatilmagandagi qiymatlar hamda parametrni e'lon qilishda ko'p yozuv imkoniyati hisoblanadi.

Ko'rsatilmagandagi qiymatlardan foydalanishdagi muammolarga doir misollar.

```
int sum(int x);
int sum(int x, int y = 2) ;
cout << sum(10, 20) << endl; // Normal
cout << sum(10) << endl; // Turlilik
```

Navbatdagi misolda berilganlarning tipini tanlash (float yoki double tip) mumkin bo'lmaganligi sababli, avtomatik butun songa almashtirish lozim:

```
float sum(float x, float y);
double sum(double x, double y);
cout << sum(10.5, 20.4) << endl; // Normal
cout << sum(10, 20) << endl; // Turlilik
```

Faqat double tipiga ega parametrlarga ega bo'lgan funksiya mavjud bo'lganda, butun son avtomatik ravishda double tipiga almashtirilgan bo'lar edi. Huddi shunday muammo bir vaqtda ishorali va ishorasiz tiplardan foydalanilganda yuzaga keladi:

```
void print(char ch);
void print(unsigned char ch);
print ('S'); // Normal
print (119); // Turlilik
```

Quyidagi misol parametrni e'lon qilishda turlicha yozish imkoniyatining mavjudligidagi muammoni namoyish qiladi (char *str e'loni char str[]ga ekvivalent):

```
void print(char *str);
void print(char str[]);
print ("String"); // Turlilik
```

Bundan tashqari, funksiyaga parametrlarni turlicha berish turlilikka olib kelishi mumkin.

```
void print(int x);
void print(int &x);
int n = 25;
print (n); // Turlilik
```

7.7. Funksiyadan qiymat qaytarish usullari

Funksiyadan qiymatni qaytarish uchun return operatoridan foydalaniladi. Bu operator bajarilganidan keyin funksiyalarning bajarilishi to'xtaydi va boshqaruv funksiyani chaqirish nuqtasiga qaytadi. Bu shuni anglatadiki, return operatoridan keyingi ko'rsatmalar hech qachon bajarilmaydi. Agar funksiya ichida return operatori bo'lmasa, funksiyani yopiluvchi qavsga kelgandan keyin boshqaruv funksiyani chaqirish nuqtasiga qaytariladi. Bunda qaytariluvchi qiymat aniqlanmaydi.

Agar funksiya hech qanday qiymat qaytarmasa, funksiya e'lonida berilganlar tipi o'rnida void kalit so'zi ko'rsatiladi. Bunday funksiya ishida return operatori bo'lmasligi mumkin, ammo undan, muddatidan oldin funksiyadan chiqish uchun, qiymatni ko'rsatmagan holda foydalanish mumkin. Hech qanday qiymat qaytarmaydigan funksiyani chaqirishni qandaydir ko'rsatma ichida joylashtirish mumkin emas. Faqat alohida ko'rsatmada. Qiymat qaytarmaydigan funksiyaga misol:

```
void print(const char *s) {
 cout << s << endl;
}
```

Boshqa hollarda funksiya e'loni va aniqlanishida funksiya nomidan oldin berilganlarni qaytaruvchi tipi beriladi. Bu tipning qiymati return operatorida ko'rsatilishi lozim. Bunda qiymatning nusxasi qaytariladi. Funksiya massivdan tashqari har qanday tipning qiymatni qaytarishi mumkin. Massivlar bilan funksiya paramertlari orqali, unga ko'rsatkichni bergan holda yoki aniq elementga ko'rsatkichni qaytargan holda ishlash zarur. Qandaydir qiymat qaytaruvchi funksiyani chaqirishni ifoda ichida "=" operatoridan keyin o'ng tomonda joylashtirish mumkin. Agar funksiya murojaat qaytarsa, funksiyani joylashtirishni "=" operatoridan chap tomonda joylashtirishga ruxsat beriladi. Qaytariluvchi qiymatni o'zgaruvchiga o'zlashtirish yoki shunchaki e'tiborga olmaslik mumkin. Qiymatni qaytaruvchi funksiyaga misol:

```
int sum(const int x, const int y) {
 return x + y;
}
```

return operatoridan foydalanishda turlicha (har xil) vaziyatlar bo'lmasligi lozim. Masalan, quyidagi holatda qaytariluvchi qiymat shartga bog'liq bo'ladi:

```
int sum(int x, int y) {
 if(x > 0) {
 return x + y;
 }
}
```

Agar «x» o'zgaruvchi noldan katta qiymatga ega bo'lsa, hammasi joyida, ammo agar o'zgaruvchi nolga teng yoki manfiy qiymatga ega bo'lsa, u holda qaytariluvchi qiymat aniqlanmagan va funksiya «мусор» deb nomlanuvchi ihtiyoriy qiymat qaytaradi. Bunday hollarda kompilyatsiya jarayonida ogohlantiruvchi xabar chiqariladi: "*warning C4715: sum:* значение возвращается не при всех путях выполнения". Bunday turlikni chetlab o'tish uchun funksiya tanasi oxirida return operatorini ko'rsatilmagandagi qiymat bilan qo'yish lozim:

```
int sum(int x, int y) {
 if(x > 0) {
 return x + y;
 }
 return 0;
}
```

Funksiya ko‘rsatkich qaytarishi mumkin. Bunday holda funksiya e’lonida va aniqlanishida ko‘rsatkichning mos tipi ko‘rsatiladi. Misol sifatida satrning oxirgi belgisiga ko‘rsatkichni yoki satr bo‘sh bo‘lsa nolinchi ko‘rsatkichni qaytaramiz.

Ko‘rsatkichni qaytarish dasturi listingi

```
#include <iostream>
#include <cstring>
char *func(char *s);
int main() {
 char *p = 0, str[] = "String";
 p = func(str);
 if(p) { // agar nol ko‘rsatkich bo‘lmasa
 cout << *p << endl; // g
 } else cout << "NULL" << endl;
 cin.get();
 return 0;
}
char *func(char *s) {
int len = strlen(s); // Satr uzunligini olamiz
 if(!len) return 0; // Bo‘sh bo‘lsa, nol ko‘rsatkich
 else return &s[len - 1]; // Oxirgi belgiga ko‘rsatkich
}
```

```
g
```

VC++ tilida funksiyalar murojaatlarni qaytarishi ham mumkin. Bunday holda funksiyani "=" operatoridan ham o‘ng, ham chap tomonda joylashtirishga ruxsat beriladi. Funksiya murojaat qaytarishi uchun berilganlar tipini e’lon qilish va aniqlashdan

keyin "&" operatori k'rsatiladi. Misol tariqasida ko'rsatilgan indeks bo'yicha massiv elementi qiymatini olish yoki o'zgartirish imkonini beruvchi funksiya yozamiz.

Murojaat qaytarish dasturi listingi.

```
#include <iostream>
int &at(int *a, int i) ;
int main() {
 int mas[] = {10, 20, 30};
 at(mas, 0) = 800;
 for(int i = 0; i < 3; ++i) {
 cout << at(mas, i) << " ";
 } // 800 20 30
 cout << endl;
 cin.get();
 return 0;
}
int &at(int *a, int i) {
 return a[i];
}
```

8. Ko'rsatkichlar va dinamik massivlar

8.1. Ko'rsatkichlar

Ko'rsatkich – bu boshqa o'zgaruvchining manzilini saqlash uchun tayinlangan o'zgaruvchi. VC++ tilida ko'rsatkichlardan quyidagi hollarda tez-tez foydalaniladi:

- dinamik xotirani boshqarish uchun;
- o'zgaruvchining qiymatini funksiya ichida o'zgartirish imkoniyatiga ega bo'lish uchun;
- massivlar va boshqalar bilan samarali ishlash uchun.

Ko'rsatkichni e'lon qilish quyidagicha formatga ega:

<Tip> *<O'zgaruvchi>;

int tipidagi ko'rsatkichni e'lon qilishga misol: int *p;

Ko'pincha "*" belgisi o'zgaruvchi nomidan oldin emas, balki tipdan keyin ko'rsatiladi: int* p;

Kompilyator uchun har ikkala e'lon hech qanday farqga ega emas. Ammo, bir ko'rsatmada bir necha o'zgaruvchini e'lon qilishda "*" belgisi o'zgaruvchilar tipiga emas, balki o'zidan keyin ko'rsatilgan o'zgaruvchiga tegishli bo'lishini hisobga olish lozim. Masalan, quyidagicha ko'rsatma ikkita ko'rsatkichni emas, ko'rsatkich va o'zgaruvchini e'lon qiladi:

int* p, x; *// "x" o'zgaruvchi ko'rsatkich emas*

Shu sababli "*" belgisini o'zgaruvchi nomidan oldin ko'rsatish maqsadga muvofiq: int *p, x;

O'zgaruvchining manzilini ko'rsatkichga o'zlashtirish uchun o'zgaruvchining nomidan oldin "&" operatori ko'rsatiladi. Bunda o'zgaruvchi va ko'rsatkichning tiplari mos kelishi lozim. Bu manzil arifmetikasini hisoblashda berilganlarning o'lchami ma'lum bo'lishi uchun zarur. Manzilni o'zlashtirishga misol:

```
int *p, x = 10;
p = &x;
```

Ko'rsatkich murojaat qilayotgan manzilda joylashgan qiymatni olish yoki o'zgartirish uchun ko'rsatkichni qayta nomlash operatoridan foydalaniladi. Buning uchun o'zgaruvchi nomidan oldin "*" operatori ko'rsatiladi. Masalan:

```
cout << *p << endl; // 10
*p = *p + 20;
cout << *p << endl; // 30
```

Ko'rsatkichlar bilan bajariladigan asosiy amallar quyidagi listingda keltirilgan.

```
#include <iostream>
int x = 10;
int *p = 0; // Nolinchi ko'rsatkich
int main() {
  p = &x; // "x" o'zgaruvchining manzilini ko'rsatkichga o'zlashtirish
```

```
// Manzilni chiqarish
cout << p << endl; // Masalan: 00A0A000
// Qiymatni chiqarish
cout << *p << endl; // 10
cin.get();
return 0;
}
```

Keltirib o'tilgan misolda ko'rsatkich e'lon qilinganda '0' qiymat o'zlashtirildi. Chunki, lokal ko'rinish sohasida e'lon qilingan ko'rsatkichlar ihtiyoriy qiymat oladi. Bunday ko'rsatkich orqali qandaydir qiymatni yozishga harakat qilish operatsion tizimning buzilishiga olib kelishi mumkin. Shu sababli, kelishuvga ko'ra hech narsani ko'rsatmaydigan ko'rsatichlar '0' qiymatga ega bo'lishi zarur. Sonli qiymat o'rniga stdio.h sarlavha faylida aniqlangan NULL makrosini ko'rsatish mumkin.

Makrosning aniqlanishi quyidagicha: #define NULL 0

Makrosdan foydalanishga misol:

int *p = NULL; // *int *p = 0; ga ekvivalent*

Bir ko'rsatkichning qiymatini boshqa ko'rsatkichga o'zlashtirish mumkin.

```
int *p1 = 0, *p2 = 0, x = 10;
p1 = &x; p2 = p1; // "x" o'zgaruvchi manzilini nusxalash
cout << *p2 << endl; // 10
*p2 = 40; // "x" o'zgaruvchining qiymatini o'zgartirish
cout << *p1 << endl; // 40
```

Yuqorida keltirib o'tilgan misolda 'x' o'zgaruvchining manzili bir ko'rsatkichdan boshqasiga nusxalandi. Manzilni nusxalashdan tashqari ko'rsatkichga ko'rsatkich yaratish ham mumkin. Buning uchun o'zgaruvchi nomidan oldin ikkita '*' operatori qo'yiladi: int **p = 0;

Ko'rsatkich manzilini olish uchun "&" operatoridan foydalaniladi, ko'rsatkich murojaat qilayotgan o'zgaruvchi qiymatini olish uchun esa ikkita "*" operatori qo'llaniladi. Masalan,

```
int *p1 = 0, **p2 = 0, x = 10;
p1 = &x; p2 = &p1; // ko'rsatkichga ko'rsatkich
cout << **p2 << endl; // 10
**p2 = 40; // "x" o'zgaruvchida qiymatni o'zgartirish
cout << *p1 << endl; // 40
```

Ko'rsatkichlarni ichma-ich joylashlar soni chegaralanmagan. har bir joylashtirishda qo'shimcha yulduzcha ko'rsatiladi.

```
int *p1 = 0, **p2 = 0, ***p3 = 0, x = 10;
p1 = &x; p2 = &p1; p3 = &p2;
cout << ***p3 << endl; // 10
***p3 = 40; // «x» o'zgaruvchida qiymatni o'zgartirish
cout << **p2 << endl; // 40
```

Bunday sintaksisni tushunish qiyin va oson xatolikka yo'l qo'yish mumkin. Shu sababli odatda murojaatga murojaatdan foydalanish bilan chegaralaniladi. Ko'rsatkichni nomlashda unga nafaqat sonli qiymat, balki satr ham o'zlashtirish mumkin. Masalan:

```
char *str = "String";
cout << str << endl; // String
```

8.2. Murojaatlar

Murojaat – boshqa o'zgaruvchining yangi nomi hisoblanuvchi o'zgaruvchi. Murojaatga faqat uni e'lon qilganda qiymat berish mumkin. Uning tipi o'zgaruvchining tipi bilan mos kelishi lozim.

Murojaatni e'lon qilish quyidagicha:

<Tip> &<Murojaat nomi> = <O'zgaruvchi>;

Murojaat yaratilganidan so'ng uning nomini ifodalarda o'zgaruvchining nomi o'rniga foydalanish mumkin.

// Bog'liq bo'lmagan murojaatlar

```
#include <iostream>
int main() {
 int x = 10;
 int &ref = x; // Murojaatni e'lon qilish
 ref = 20;
 cout << x << endl; // 20
 x = 30;
 cout << ref << endl; // 30
 cin.get();
 return 0;
}
```

```
20
30
```

Murojaatlardan foydalanishning qulayligi parametrlarni funksiyaga berishda ko'rinadi. Murojaat orqali parametrlarni uzatishni ko'rishdan avval, ko'rsatkichlardan foydalanish bilan tanishib chiqamiz.

```
// Parametrlarni murojaat bo'yicha uzatish
#include <iostream>
void func(int *x);
int main() {
 int y = 10;
 func(&y); // Manzil uzatiladi, qiymat emas
 cout << y << endl; // 20
 cin.get();
 return 0;
}
void func(int *x) {
 *x = *x * 2;
}
```

Keltirib o'tilgan misolda ichida o'zgaruvchining qiymatini ikki marta kattalashtirishni amalga oshiradigan func() funksiyasi e'lon qilingan. Qiymat bo'yicha uzatishdan (ko'rsatilmaaganda) foydalanilsa, u holda qiymat nusxasi yaratiladi va barcha amallar

nusxa bilan bajariladi. Lokal o'zgaruvchilar faqat funksiya ichida ko'ringani uchun funksiya bajarilganidan keyin o'chiriladi. Nusxa qiymatining ihtiyoriy o'zgarishi originalning qiymatiga ta'sir qilmaydi. O'zgaruvchining qiymatini o'zgartirish uchun qiymatni murojaat orqali uzatishdan foydalaniladi. Bunday holda funksiyaga o'zgaruvchining manzili uzatiladi: func (&y);

Funksiyaning ichida manzil ko'rsatkichga o'zlashtiriladi. Ko'rsatkich-ni qayta nomlash amalidan foydalangan holda o'zgaruvchining qiymatini o'zgartirish mumkin, nusxasining qiymatini emas. Bu quyidagicha ko'rsatma orqali amalga oshiriladi: *x = *x * 2;

Murojaatdan foydalanilganda '&' operatori funksiya e'lonida parametrdan oldin ko'rsatiladi. Bunday holda funksiya chaqirilganda manzil emas, qiymat uzatiladi. Shunday qilib, funksiya ichida ko'rsatkich bilan emas, balki o'zgaruvchining laqabi (o'zgartirilgan nomi) bilan ishlanadi. Shu sababli, ko'rsatkichni qayta nomlash shart emas.

Quyida murojaatlardan foydalanishga misol keltirilgan.

```
// Parametrlarni murojaat yordamida uzatish
#include <iostream>
void func(int &x);
int main() {
 int y = 10;
 func(y); // Qiymat uzatiladi, manzil emas
 cout << y << endl; // 20
 cin.get();
 return 0;
}
void func(int &x) {
 x = x * 2;
}
```

x=x*2; ko'rsatmasi *x=*x*2; ko'rsatmaga nisbatan yaxshiroq ko'rinadi.

8.3. Xotirani dinamik ajratish

O‘zgaruvchi e’lon qilinganida berilganlar tipini ko‘rsatish, massiv uchun esa qo‘shimcha ravishda elementlarning aniq sonini berish zarur. Dastur ishga tushirilganda bu axborot asosida zarur xotira hajmi avtomatik ravishda ajratiladi. Dastur ishi tugallanganidan so‘ng xotira avtomatik bo‘shaydi. Boshqa so‘z bilan aytganda, dastur bajarilguncha xotira hajmimi bilish zarur. Dasturning bajarilishi jarayonida yangi o‘zgaruvchi yaratish yoki mavjud massiv o‘lchamini o‘zgartirish mumkin emas.

Dasturning bajarilishi jarayonida massiv o‘lchamining kattalashishini amalga oshirish uchun new operatori yordamida xotiradan yetarlicha hajm ajratish zarur. Mavjud elementlarni ko‘chirib, so‘ngra yangi elementlarni qo‘shish mumkin. Dinamik xotirani boshqarish dasturchiga yuklatilgan. Shu sababli, xotira bilan ishlashni tugallagandan so‘ng delete operatori yordamida uni operatsion tizimga qaytarish zarur. Agarda xotira qaytarilmasa, u keyinchalik foydalanish uchun berk bo‘ladi. Bunday holatlar xotiraning yo‘qolishiga olib keladi. Ular kirishli bo‘lishi uchun kompyuterni qayta yuklash lozim bo‘ladi.

Quyidagi sintaksis bir obyekt uchun xotira ajratishga mo‘ljallangan:

<Ko‘rsatkich> = new<Berilganlar tipi>(<Boshlang‘ich qiymat>);

new operatori ko‘rsatilgan tip qiymatlarini saqlash uchun zarur bo‘lgan xotira hajmini ajratadi, bu xotiraga boshlang‘ich qiymatni yozadi (agar berilgan bo‘lsa) va manzilni qaytaradi. Keyinchalik xotiraning bu qismi bilan ko‘rsatkich orqali ishlash mumkin.

Xotira ajratishga misol keltirib o‘tamiz

int *p = new int; *// Boshlang‘ich qiymatsiz*

int *p = new int(10); *// Boshlang‘ich qiymat bilan*

Xotira ajratishda xotiraning yetishmasligi yuzaga kelishi mumkin. C++ ning avvalgi talqinlarida bunday hollarda nolin-

chi ko'rsatkich qaytarilgan. Standartning yangi talqiniga ko'ra, xatolik yuzaga kelganda new operatori bad_alloc istesnosini (istesno obyekti new faylida aniqlangan) ishga tushirishi lozim. Bu istesnoga try...catch ko'rsatmasi orqali ishlov berish mumkin. Istesnoga ishlov berish bilan xotira ajratishga misol:

```
#include <new>
// ... Dastur qismi ...
int *p = 0; // Ko'rsatkich yaratish
try {
 p = new int; // Xotira ajratish
}
catch(bad_alloc err) {
 // Istesnoga ishlov berish
}
```

Xotirani ajratish try bloki ichida amalga oshiriladi. Bunda agarda bad_alloc istesnosi yuzaga kelsa, boshqaruv catch blokiga uzatiladi. catch blokidagi ko'rsatmalar bajarilganidan so'ng, boshqaruv blokdan keyin joylashgan ko'rsatmaga uzatiladi. Ya'ni, kompilyator istesnoga ishlov berildi deb qabul qiladi va dasturning bajarilishini davom ettiriladi. Ishlov berilganidan keyin ko'rsatkichdan foydalanish mumkin emas. Shu sababli odatda catch bloki ichida xatolik haqida xabar chiqariladi va dasturning bajarilishi tugallanadi. Agar istesnoga ishlov berilmasa, dastur "xalokatli" tugaydi. Iistesno yuzaga kelmasa, catch bloki ko'rsatmasi bajarilmaydi.

Ko'rsatkichni e'lon qilish try blokidan tashqarida amalga oshiriladi. Agar e'lon blok ichida bo'lsa, o'zgaruvchining ko'rinish sohasi bu blok bilan chegaralanib qoladi. Blokdan chiqilganda o'zgaruvchi avtomatik yo'qotiladi, operatsion tizimda ajratilgan xotira esa qaytarilmaydi. Shu sababli, ko'rsatkichni e'lon qilish blokdan oldin turishi kerak.

Avval ajratilgan xotirani operatsion tizimga qaytarish uchun delete operatoridan foydalaniladi. Operator quyidagicha formatga ega:

delete <Ko'rsatkich>;

delete operatori bajarilganidan keyin ko'rsatkich avvalgi qiymatini olib turadi. Shu sababli delete operatiridan foydalanilgandan keyin ko'rsatkichni tozalash (nol qiymat o'zlashtirish) qabul qilingan.

Bir obyekt uchun xotirani dinamik ajratish misoli listingi.

```
#include <iostream>
#include <cstdlib> // exit()
#include <new> // bad_alloc
int main() {
int *p = 0; // ko'rsatkich yaratish
try {
p = new int; // Xotira ajratish
}
catch(bad_alloc err) {
cout << "Xato" << endl;
exit(1); // Xatolikda chiqish
}
*p = 20; // Xotiradan foydalanish
cout << *p << endl; // 20
delete p; // Xotirani qaytarish
p = 0; // Ko'rsatkichni tozalash
cin.get();
return 0; }
```

8.4. Dinamik massivlar

Massivni e'lon qilishda elementlar sonini aniq ko'rsatish lozim. Dastur ishga tushirilganda bu axborot asosida zarur xotira hajmi avtomatik ravishda ajratiladi. Dasturning bajarilishi davrida joriy massivning o'lchamini o'zgartirish mumkin emas, ya'ni massivning o'lchami bajarilishidan avval bilish kerak. Dasturning bajarilishi jarayonida massivning kattalashishini amalga oshirish uchun new operatori yordamida xotiradan yetarlicha

xotira hajmini ajratish, mavjud elementlarni ko'chirish, so'ngra yangi elementlarni qo'shish zarur. Dinamik xotirani boshqarish dasturchiga yuklatilgan. Shu sababli xotira bilan ishlash tugallanganidan so'ng delete operatori yordamida xotirani operatsion tizimga qaytarish zarur. Agar xotira operatsion tizinga qaytarilmasa, xotira qismi keyingi foydalanishlar uchun berk bo'ladi. Bunday vaziyatlar xotiraning yo'qolishiga olib keladi. Ularni tozalash uchun kompyuterni qayta yuklash lozim bo'ladi.

Massiv uchun xotira ajratish quyidagicha ko'rinishga ega:

<Ko'rsatkich> = new<Berilganlar tipi>[<Elementlar soni>];

Ajratilgan xotirani tozalash quyidagicha bajariladi:

delete[]<Ko'rsatkich>;

Xotira tozalanganda elementlar soni ko'rsatilmaydi.

Massiv uchun xotira ajratishda xotiraning yetishmaslik holati yuzaga kelishi mumkin. Bunday hollarda C++ning avvalgi talqinlarida nol qiymat qaytarilgan. Standartning yangi talqiniga ko'ra, hatolik yuzaga kelganda new operatori bad_alloc (istesno obyekti new faylida aniqlangan) istesnosini ishga tushirishi lozim. Istesnoga ishlov berishni try...catch konstruksiyasi orqali amalga oshirish mumkin.

Istesnoga ishlov berish bilan xotirani ajratishga misol:

```
#include <new>
// ... Dastur qismi ...
const short SIZE = 3; // Massiv o'lchami
int *p = 0; // Ko'rsatkich yaratish
try {
 p = new int[SIZE]; // Xotira ajratiladi
}
catch (bad_alloc err) {
 cout << "Xato" << endl;
 exit(1); // Xatolikda chiqish
}
p[0] = 1; // Xotiradan foydalaniladi
```

```
P[1] = 2;
p[2] = 3;
for(int i = 0; i < SIZE; i++) {
 cout << p[i] << endl;
}
delete[]p; // Xotirani qaytarish (tozalash)
p = 0; // Ko'rsatkich nolga aylantiriladi
```

delete operatoridan foydalangandan keyin ham ko'rsatkich avvalgi manzilni olib turadi, shu sababli ko'rsatkichni tozalash qabul qilingan: p = 0;

8.5. Ko'rsatkich orqali massiv elementlariga kirish

Satrlar massivini e'lon qilish va qiymatlarini chiqarish quyidagicha ko'rinishga ega:

```
const char *str[] = {"String1", "String2", "String3"};
cout << str[0] << endl; // String1
cout << str[1] << endl; // String2
cout << str[2] << endl; // String3
```

Massiv elementlariga murojaat qilish uchun ko'rsatkichlardan tez-tez foydalaniladi, chunki massiv arifmetikasi indeks bo'yicha kirishga nisbatan samaraliroq. Misol tariqasida uch elementdan tashkil topgan massiv yaratamiz va uning qiymatlarini chiqaramiz.

```
#include <iostream>
int main() {
const short SIZE = 3;
int i, *p = 0, mas[SIZE] = {10, 20, 30};
// Ko'rsatkich massivning birinchi elementiga qo'yiladi
p = mas; // "&" operatori ko'rsatilmaydi!
for(i = 0; i < SIZE; i++) {
 cout << *p << endl;
```

```
    ++p; // Ko'rsatkichni navbatdagi elementiga ko'chirish
  }
  p = mas; // Ko'rsatkich holati tiklanadi
  // Qandaydir ko'rsatmalar bajariladi
  cin.get();
  return 0;
}
```

main() funksiyasi ichida birinchi satrda massiv elementlari soni saqlanadigan SIZE o'zgarmasi e'lon qilinadi. Agar massivdan tez-tez foydalanilsa, uning o'lchamini o'zgarmas sifatida ko'rsatgan ma'qul. Agar har bir siklda aniq son ko'rsatiladigan bo'lsa, massiv o'lchami o'zgarganda barcha sikllarda qo'lda o'zgartirish kiritib chiqishga to'g'ri keladi. O'zgarmas ko'rsatilganda nomlashda uning qiymatini o'zgartirish yetarli bo'ladi. Navbatdagi satrda for sikli ichida foydalanishga mo'ljallangan "i" o'zgaruvchisi e'lon qilinadi.

Bu o'zgaruvchini siklning birinchi parametrida e'lon qilish mumkin. bunda o'zgaruvchi faqat for siklining ichida ko'rinadi. Misol:

```
for(int i = 0; i < SIZE; i++) { /* Ko'rsatmalar */ }
```

Undan keyin int tip va massivga ko'rsatkich e'lon qilinadi. Massiv elementlari soni SIZE o'zgarmasi orqali beriladi. Massivni e'lon qilishda boshlang'ich qiymatlar bilan nomlnadi.

Massiv elementlarini saralash uchun for siklidan foydalaniladi. Siklning birinchi parametrida boshlang'ich qiymat (i = 0), ikkinchisida esa shart (i < SIZE), uchinchi parametrda – siklning har bir takrorlanishida bir birlikka ortishi (++i) beriladi. Sikl ichidagi ko'rsatmalar shart rost bo'lganda bajariladi ('i' o'zgaruvchining qiymati massiv elementlari sonidan kichik).

O'zgaruvchilar e'lon qilinganidan keyin ko'rsatkichga massiv birinchi elementining manzili o'zlashtiriladi. Massiv nomidan oldin "&" operatori ko'rsatilmaydi, chunki o'zgaruvchining nomi birinchi element manzilini oladi. "Agar &" operatoridan

foydalanilsa, u holda qo'shimcha ravishda kvadrat qavslar ichida indeksni ko'rsatish zarur bo'ladi.

```
p = &arr[0]; // p = mas; ga ekvivalent
```

Demak, o'zgaruvchining nomi massivning birinchi elementiga murojaat qiluvchi ko'rsatkich hisoblanadi. Shu sababli massivning elementini chaqirish kvadrat qavslar ichida ko'rsatilgan indeks bo'yicha yoki manzil arifmetikasidan foydalanish orqali amalga oshirilishi mumkin. Quyidagi chiqarish ko'rsatmalari teng kuchli hisoblanadi:

```
int mas[3] = {1, 2, 3};
cout << mas[1] << endl; // 2
cout << *(mas+1) << endl; // 2
cout << *(1+mas) << endl; // 2
cout << 1[mas] << endl; // 2
```

Oxirgi ko'rsatma g'alati ko'rinishi mumkin. Ammo, 1[mas] ifoda kompilyator tomonidan *(1+mas) kabi qabul qilinishini e'tiborga oladigan bo'lsak, hammasi joyida bo'ladi.

Kvadrat qavslarda indekslarni ko'rsatishda har safar massivning mos elementini hisoblash bajariladi. Bu jarayon samarali bo'lishi uchun ko'rsatkich e'lon qilinadi va unga birinchi elementning manzili o'zlashtiriladi. Elementga kirish uchun ko'rsatkich mos elementga suriladi. Ko'rsatkichni e'lon qilish va unga massivning birinchi elementi manzilini o'zlashtirishga misol:

```
int *p = 0;
int mas[3] = {1, 2, 3};
p = mas;
```

O'zgaruvchining nomi birinchi element manzilini olgani uchun massiv nomidan oldin "&" operatori ko'rsatilmaydi. "&" operatoridan foydalanilsa, kvadrat qavs ichida qo'shimcha ravishda indeksni ko'rsatish zarur:

```
p = &mas[0]; // Ekvivalenti: p=mas;
```

Huddi shunday ko'rsatkichga massivning ihtiyoriy elementini o'zlashtirish mumkin, masalan, uchinchi elementini:

```
p = &mas[2]; // Uchinchi elementga ko'rsatkich
```

Ko'rsatkich murojaat qilayotgan element qiymatini olish uchun qayta nomlash (разыменования) amali bajarilishi zarur. Buning uchun ko'rsatkich nomidan oldin "*" operatori qo'shiladi. Masalan:

```
cout << *p << endl;
```

Ko'pgina hollarda ko'rsatkichlardan massiv elementlarini saralashda foydalaniladi. Misol tariqasida uch o'lchamli massiv elementlarini for operatori yordamida chiqarishni ko'ramiz.

```
const short SIZE = 3;
int *p = 0, mas[SIZE] = {1, 2, 3};
// Ko'rsatkich massivning birinchi elementiga qo'yiladi
p = mas; // "&" operatori ko'rsatilmaydi!!!
for(int i = 0; i < SIZE; i++) {
 cout << *p << endl;
 ++p; // Ko'rsatkich navbatdagi elementga ko'chiriladi
 }
p = mas; // Ko'rsatkichning holati tiklanadi
```

Dasturning birinchi satrida massivdagi elementlar sonini saqlovchi SIZE o'zgarmasi e'lon qilingan. Agar massivdan tez-tez foydalanilsa, uning o'lchamini o'zgarmas sifatida saqlagan ma'qul, chukni massivni har bir saralashda uni ko'rsatishga to'g'ri keladi. Agar har bir siklda aniq son ko'rsatilsa, massiv o'lchami o'zgarganda barcha sikllarda uni qo'lda o'zgartirib chiqishga to'g'ri keladi. O'zgarmas e'lon qilinganida uning qiymatini bir marta, e'lon davrida, o'zgartirish yetarli bo'ladi.

Navbatdagi satrda int tipiga ko'rsatkich va massiv e'lon qilindi. Massiv elementlari soni SIZE o'zgarmasi orqali beriladi. E'lon qilishda massiv boshlang'ich qiymatlari bilan nomlanadi.

for sikli ichida ko'rsatkich murojaat qilayotgan element qiymati chiqariladi, so'ngra ko'rsatkich qiymati bir birlikka (++p;) ortadi. Bunda massivning elementi emas, balki manzili o'zgaradi. Ko'rsatkich qiymati kattalashganda oddiy arifmeti-

ka qoidasi emas, balki manzilli arifmetika qoidasi qo'llaniladi. Ko'rsatkich qiymatining bittaga ortishi qiymatning tip o'lchamiga ortishini bildiradi. Masalan, agar int tipi 4 bayt band qilsa, qiymat bittaga ortganda ko'rsatkich manzili '0x0012FF30' o'rniga '0x0012FF34' manzilini oladi. Qiymat '1'ga emas '4'ga ortdi. Yuqoridagi misolda sikl ichida ikkita ko'rsatma o'rniga bitta ko'rsatmadan foydalanish mumkin:

```
cout << *p++ << endl;
```

p++ ifoda joriy manzilni qaytaradi va bir birlikka ortadi. "*" belgisi ko'rsatilgan manzil bo'yicha element qiymatiga kirish imkonini beradi. Bajarilishi qavslarni qo'yilishng quyidagi ketma-ketligiga mos:

```
cout << *(p++) << endl;
```

Agarda qavslar quyidagicha qo'yilsa:

```
cout << (*p)++ << endl;
```

u holda, massiv elementiga kiriladi va uning joriy qiymati chiqariladi, so'ngra massiv elementi qiymati bittaga orttiriladi. Ko'rsatkichning navbatdagi elementga ko'chishi amalga oshirilmaydi.

for siklida ko'rsatkichdan o'zgaruvchi-hisoblagich sifatida foydalanish mumkin. Bu holda boshlang'ich qiymat massivning birinchi elementi manzili bo'ladi, davom etish sharti esa – birinchi element manzilidan kichik manzil plyus elementlar soni. Orttirma oddiy kabi amalga oshiriladi, faqat oddiy arifmetika o'rniga manzilli arifmetika qo'llaniladi. Misol:

```
const short SIZE = 3;
int mas[SIZE] = {1, 2, 3};
for(int *p = mas; p < mas + SIZE; p++) {
 cout << *p << endl;
}
```

Massivning barcha elementlarini while sikli va ko'rsatkich yordamida quyidagicha chiqarish mumkin:

```
const short SIZE = 3;
```

```
int *p = 0, i = SIZE, mas[SIZE] = {1, 2, 3};
p = mas;
while(i-- > 0) {
 cout << *p++ << endl;
}
p = mas;
```

Ko'rsatkichlar massivi. Ko'rsatkichlarni massivda saqlash mumkin. Ko'rsatkichlar massivining e'lonida quyidagi sintaksisdan foydalaniladi:

<Tip> *<Massiv nomi>[<Elementlar soni>];

Ko'rsatkichlar massividan foydalanishga sodda misol:

```
int *p[3]; // Uch elementli ko'rsatkichlar massivi
int x = 10, y = 20, z = 30;
p[0] = &x; p[1] = &y; p[2] = &z;
cout << *p[0] << endl; // 10
cout << *p[1] << endl; // 20
cout << *p[2] << endl; // 30
```

8.6. Funksiyaga ko'rsatkichlar

Funksiyalar ham o'zgaruvchi kabi ko'rsatkichda saqlash uchun manzilga ega. Keyinchalik ko'rsatkich orqali bu funksiyani chaqirish mumkin. Bundan tashqari, funksiyaga ko'rsatkichni boshqa funksiyaning parametri sifatida uzatish mumkin. Funksiyaga ko'rsatkichni e'lon qilish quyidagicha ko'rinishga ega:

<Tip>(*<Ko'rsatkich nomi>)([<Tip1>[,..., <TipN>]]);

Ko'rsatkichga funksiya manzilini o'zlashtirish uchun funksiya nomini parametrlarsiz va «=» operatoridan o'ng tomonda qavs ichida ko'rsatiladi. Bunda funksiya va ko'rsatkich parametrlari tipi mos kelishi lozim. Ko'rsatkichni e'lon qilish va funksiya manzilini o'zlashtirishga misol:

```
int (*p)(int, int);
p = sum;
```

Funksiyani ko'rsatkich orqali ikki usulda chaqirish mumkin:

```
x = (*p)(10, 20); //Huddi shunday
x = sum(10, 20); y = p(30, 10);
```

Quyidagi misolda ko'rsatkichni e'lon qilish, ko'rsatkich orqali funksiyani chaqirish va ko'rsatkichni funksiya parametri sifatida berishga misollar listingi keltirilgan.

```
// Funksiyaga ko'rsatkichlar
#include <iostream>
int sum(int x, int y);
int func(int x, int y, int (*p) (int, int));
int main() {
 int (*p)(int, int); // Funksiyaga ko'rsatkich e'loni
 p = sum; // Funksiya manzilini o'zlashtirish
 cout << (*p)(10, 20) << endl; // 30
 cout << p(30, 10) << endl; // 40
 cout << func(5, 10, sum) << endl; // 15
 cin.get();
 return 0;
}
int sum(int x, int y) {
 return x + y;
}
int func(int x, int y, int (*p)(int, int)) {
 return (*p)(x, y);
}
```

```
30
40
15
```

9. Satrlar va tuzilmalar

VC++ tilida bir baytli satrlarning ikki tipi mavjud: satr – C tilidagi o'zgaruvchi (ko'pincha C-satr deyiladi) va string sinfi. C-satr oxirgi elementi nol belgini (\0) olgan massivlar satri (char tipida) hisoblanadi. Nol belgining (nolinchi bayt) '0' soniga aloqasi yo'q. Ularning kodlari turlicha. string sinfi kirish uchun

ancha qulay interfeysni ifodalaydi. Dasturchining belgili massiv o'lchasmini nazorat qilishdan xalos etadi va satrlarga ishlov berishning ko'plab usullarni taqdim etadi. Zarurat yuzaga kelganda string sinfi obyektini C-satrga almashtirish mumkin.

9.1. Satrlar (C-satrlar)

Belgini saqlash uchun o'zgaruvchilarning char tipidan foydalaniladi. char tipiga ega bo'lgan o'zgaruvchiga sonli qiymat (belgining kodi) berish yoki belgini apostroflar ichida ko'rsatish mumkin.

Qo'shtirnoqdan foydalanish mumkin emas, bunday holda bir belgi o'rniga ikkita belgi bo'ladi: belgining o'zi va nol belgi. Misollar ko'raylik:

```
char ch1, ch2;
ch1 = 119; // w harfi
ch2 = 'w'; // w harfi
```

Apostroflar ichida maxsus belgilarni ko'rsatish mumkin. Sakkizlik va o'n oltilik qiymatlarni ko'rsatishga misol:

```
char ch1, ch2;
ch1 = '\167'; // w harfi(sakkizlik qiymati)
ch2 = '\x77'; // w harfi(o'n oltilik qiymati)
```

Ko'rsatilmaganda char tipi ishorali hisoblanadi va -128 dan 127 gacha oraliqdagi qiymatlarni saqlash imkonini beradi. Agar tipdan oldin unsigned kalit so'zi ko'rsatilsa, oraliq '0'dan '255'gacha bo'ladi. O'zgaruvchiga belgi o'zlashtirilganida u avtomatik tarzda mos butun kodga almashtiriladi.

char tipi xotirada bir bayt (8 bit) band qiladi. Tip ishorali bo'lsa, u holda yuqori bit ishora belgisini oladi: '0' qiymat nomanfiy songa, '1' manfiy songa mos keladi. Agar tip ishorasiz bo'lsa, ishora belgisidan foydalanilmaydi. Buni ishorali tipni ishorasiz tipga almashtirishda hisobga olish lozim, yuqori bit katta qiymatning hosil bo'lishiga sabab bo'ladi:

```
char ch1 = -1;
unsigned char ch2 = (unsigned char)ch1;
```

cout << (int)ch2 << endl; // 255

Kodi '128'dan (7 bit) kichik bo'lgan belgilar ASCII kodlashiga mos keladi. Bu belgilarning kodlari barcha bir baytli kodlashlarda bir xil. ASCII kodlash tarkibiga raqamlar, lotin alifbosi harflari, tinish belgilari va ba'zi bir xizmatchi belgilar (satrni ko'chirish, tabulyatsiya va boshqalar) kiradi.

unsigned char tipining sakkizinchi biti milliy alifbo belgilarini kodlash uchun mo'ljallangan. char tipida bu belgilar manfiy qiymatga ega (yuqori bit ishora belgisini oladi). Demak, char tipi hammasi bo'lib '256' belgini kodlash imkonini beradi.

Rus alifbosi harflarini kodlash uchun beshta kodlash jadvali tayinlangan: Windows-1251 (cp1251), cp866, iso8859-5, koi8-r va mac-cyrillic. Bu kodlashlarda ayni bir rus harfining kodi turlicha bo'lishi mumkin.

Belgi registrini o'zgartirish. C++ tilida belgi registrini o'zgartirish uchun quyidagi funksiyalar tayinlangan:

toupper() funksiyasi belgi kodini yuqori regisrda qaytaradi. Reristrni almashtirish amalga oshirilmasa, belgi kodi o'zgarishsiz qaytariladi. Funksiyaning prototipi:

```
#include <cctype>
int toupper(int C);
```

Misol:

```
setlocale(LC_ALL,"Russian_Russia.1251");
cout << (char)toupper('w') << endl; // W
cout << (char)toupper('W') << endl; // W
cout << (char)toupper((unsigned char)'б') << endl; // Б
```

tolower() funksiyasi belgi kodini quyi regisrda qaytaradi. Agar reristrni almashtirish amalga oshirilmasa, belgi kodi o'zgarishsiz qaytariladi. Funksiyaning prototipi:

```
#include <cctype>
int tolower(int C);
```

Misol:

```
setlocale(LC_ALL,"Russian_Russia.1251");
```

```
cout << (char)tolower('w') << endl; // w
cout << (char)tolower('W') << endl; // w
cout << (char)tolower((unsigned char) 'Б') << endl; // б
```

Belgi tipini aniqlash. Olingan belgi tipini aniqlash uchun <cctype> kutubxonasining quyidagi funksiyalardan foydalaniladi:

isdigit() funksiyasi belgi o'nlik raqam bo'lsa, nol bo'lmagan qiymat, aks nolda '0' qiymat qaytaradi. Funksiyaning prototipi quyidagicha:

```
#include <cctype>
int isdigit(int C);
```

Masalan:

```
cout << isdigit('w') << endl; // 0
cout << isdigit('2') << endl; // 4
cout << isdigit((unsigned char)'б') << endl; // 0
```

isxdigit() funksiyasi qiymat o'n oltilik raqam bo'lsa, nol bo'lmagan qiymat qaytaradi (0 dan 9 gacha raqam yoki A dan F gacha harf (registr ahamiyatga ega emas), aks holda – 0. Funksiyaning prototipi:

```
int isxdigit(int C);
```

Masalan:

```
cout << isxdigit('8') << endl; // 128
cout << isxdigit('a') << endl; // 128
cout << isxdigit('F') << endl; // 128
cout << isxdigit('б') << endl; // 0
```

isalpha() funksiyasi belgi harf bo'lsa, nol bo'lmagan qiymat qaytaradi, aks holda – 0. Funksiyaning prototipi: int isalpha(int C);

Rus harflari uchun lokalni zoslash zarur bo'ladi. Masalan:

```
setlocale(LC_ALL,"Russian_Russia.1251");
cout << isalpha('w') << endl; // 258
cout << isalpha('2') << endl; // 0
cout << isalpha((unsigned char)'б') << endl; // 258
cout << isalpha((unsigned char)'Б') << endl; // 257
```

isspace() funksiyasi belgi bo'shliq belgisi (bo'shliq, tabulyatsiya, satrni ko'chirish yoki karetkani qaytarish) bo'lsa, nol bo'lmagan qiymat qaytaradi, aks holda – 0. Funksiyaning prototipi: int isspace(int C);

Masalan:

```
cout << isspace('w') << endl; // 0
cout << isspace('\n') << endl; // 8
cout << isspace('\t') << endl; // 8
cout << isspace((unsigned char)'б') << endl; // 0
```

isalnum() funksiyasi belgi harf yoki raqam bo'lsa, nol bo'lmagan qiymat, aks holda – 0 qiymat qaytaradi. Funksiyaning prototipi:

int isalnum(int C);

Rus harflari uchun lokalni zoslash zarur bo'ladi. Masalan:

```
setlocale(LC_ALL,"Russian_Russia.1251");
cout << isalnum('w') << endl; // 258
cout << isalnum('8') << endl; // 4
cout << isalnum('\t') << endl; // 0
cout << isalnum((unsigned char)'б') << endl; // 258
```

islower() funksiyasi belgi quyi registrdagi harf bo'lsa, nol bo'lmagan qiymat qaytaradi, aks holda – 0. Funksiyaning prototipi:

int islower(int C);

Rus harflari uchun lokalni zoslash zarur bo'ladi. Masalan:

```
setlocale(LC_ALL,"Russian_Russia.1251");
cout << islower('w') << endl; // 2
cout << islower('8') << endl; // 0
cout << islower('\t' ) << endl; // 0
cout << islower((unsigned char)'б') << endl; // 2
cout << islower((unsigned char)'Б') << endl; // 0
```

isupper() funksiyasi belgi yuqori registrdagi harf bo'lsa, nol bo'lmagan qiymat qaytaradi, aks holda – 0. Funksiyaning prototipi:

int isupper(int C);

Rus harflari uchun lokalni zoslash zarur bo'ladi. Masalan:

```
setlocale(LC_ALL,"Russian_Russia.1251");
cout << isupper('W') << endl; // 1
cout << isupper('8') << endl; // 0
cout << isupper('\t') << endl; // 0
cout << isupper((unsigned char)'б') << endl; // 0
cout << isupper((unsigned char)'Б') << endl; // 1
```

ispunct() funksiyasi belgi *punktatsiya* belgisi bo'lsa, nol bo'lmagan qiymat qaytaradi, aks holda – 0. Funksiyaning prototipi:

int ispunct(int C);

Masalan:

```
cout << ispunct('8') << endl; // 0
cout << ispunct('f') << endl; // 0
cout << ispunct('!') << endl; // 16
cout << ispunct('\') << endl; // 16
cout << ispunct(1) << endl; // 0
```

isprint() funksiyasi belgi chop qiliniuvchi (bo'shliq ham) belgi bo'lsa, nol bo'lmagan qiymat qaytaradi, aks holda – 0. Funksiya prototipi:

int isprint(int C);

Rus harflari uchun lokalni zoslash zarur bo'ladi. Masalan:

```
setlocale(LC_ALL,"Russian_Russia.1251");
cout << isprint('8') << endl; // 4
cout << isprint('\x5') << endl; // 0
cout << isprint('1') << endl; // 4
cout << isprint((unsigned char)'б') << endl; // 258
```

isgraph() funksiyasi belgi chop qiliniuvchi (bo'shliq chop qilinuvchi emas) belgi bo'lsa, nol bo'lmagan qiymat qaytaradi, aks holda – 0. Funksiyaning prototipi: int isgraph(int C);

Rus harflari uchun lokalni zoslash zarur bo'ladi. Masalan:

```
setlocale(LC_ALL,"Russian_Russia.1251");
```

```
cout << isgraph('8') << endl; // 4
cout << isgraph('\x5') << endl; // 0
cout << isgraph(1) << endl; // 0
cout << isgraph((unsigned char)'б') << endl; // 258
```

iscntrl() funksiyasi belgi chop qilinmaydigan belgi bo'lsa, nol bo'lmagan, aks holda – 0 qiymat qaytaradi. Funksiyaning prototipi:

```
int iscntrl(int C);
```

Misol:

```
cout << iscntrl('8') << endl; // 0
cout << iscntrl('\x5') << endl; // 32
cout << iscntrl(' ') << endl; // 0
```

Rus harflaridan oldin unsigned char tipga keltirish amali ko'rsatiladi. Agarda tipga keltirish ko'rsatilmasa, unsigned int tipiga almashtirish bajariladi. Rus harflari manfiy qiymatga ega bo'lgani uchun ishora biti, char tipi diapozonidan chiqib ketuvchi, katta qiymatlarning chiqishiga sababchi bo'ladi. Masalan, "б" rus harfi uchun qiymat 4294967265 ga teng bo'ladi.

```
cout << (unsigned int)(unsigned char)'б' << endl; // 225
cout << (unsigned int)'б' << endl; // 4294967265
```

C-satrlar. C-satr oxirgi elementi nol belgini (\0) olgan belgilar massivi (char tipida) hisoblanadi. C-satr elementlari char tipida bo'lgan massiv kabi e'lon qilinadi: char str[7];

C-satr nomlanganda figurali qavslar ichida belgilarni sanab o'tish

```
char str[7] = {'S', 't', 'r', 'i', 'n', 'g', '\0'};
```

yoki satrni qo'shtirnoq ichida ko'rsatish mumkin:

```
char str[7] = "String";
```

Qo'shtirnoqdan foydalanilganda satr uzunligi bir belgiga uzun bo'lishini e'tiborga olish lozim. Chunki, satrning oxiriga avtomatik ravishda nol belgi qo'yiladi. E'lon qilish jarayonida massivning o'lchami ko'rsatilmasa, satr uzunligiga mos ravishda avtomatik aniqlanadi.

```
char str[] = "String";
```

Qo'shtrirnoq ichida qiymat berish faqat nomlash jarayonida amalga oshiriladi. Keyinchalik satrga o'zlashtirishga harakat qilish xatolikka olib keladi:

```
char str[] = "String";
char str[7];
str = "String"; // Xato !
```

Satr ichida qo'shtirnoq ichida maxsus belgilarni ko'rsatish mumkin (masalan, \n, \r va h.k.). Agar satr ichida qo'shtirnoq uchrasa, u holda uni teskari slesh (og'ma chiziq) yordamida ekranlashtirish lozim:

```
char str[] = "Guruh\"Kino\"\n";
```

Agarda satr ichida teskari slesh (\) uchrasa, uni ekranlashtirish lozim. Buni faylga yo'l ko'rsatishda e'tiborga olish zarur:

```
char str1[] = "C:\\temp\\new\\file.txt"; // To 'g 'ri
char str2[] = "C:\temp\new\file.txt"; // Noto 'g 'ri
```

Ikkinchi satrda uchta maxsus belgilar: \t, \n va \f. almashtirilganidan keyin yo'l quyidagicha ko'rinishga ega bo'ladi:

C:<Tabulyatsiya> emp <Satrni ko'chirish> ew <Formatni ko'chirish> ile.txt

C-satrni nomlashda bir necha satrga joylash mumkin emas, bunda yangi satrga o'tish sintaktik xatolikni chaqiradi:

```
char str[] = "string1
string2"; // Ошибка: error C2001: newline в константе
```

C-satrni bir necha satrda ifodalash uchun satrni ko'chirish belgisidan oldin "\" belgisini ko'rsatish lozim:

```
char str[] = "string1\string2\string3";
// "\" belgisidan keyin hech qanday belgi bo'lmasligi lozim
cout << str << endl;
```

Agar satrlar bir ko'rsatma ichida ketma-ket joylashgan bo'lsa u holda ular bitta katta satrga birlashadi.

```
char str[] = "string1" "string2" "string3";
cout << str << endl; // string1string2string3
```

Satrlar massivini e'lon qilish quyidagicha ko'rinishga ega:

```
char str[][20] = {"String1", "String2", "String3"};
// yoki char *str[] = {"String1", "String2", "String3"};
cout << str[0] << endl; // String1
cout << str[1] << endl; // String2
cout << str[2] << endl; // String3
```

Satrni kiritish uchun getline() funksiyasidan foydalanish zarur. Birinchi parametrda cin obyekti, ikkinchi parametrda string sinfi obyekti, uchinchi parametrda esa sanash ungacha davom ettiriladigan belgi ko'rsatiladi. Agar uchinchi parametr ko'rsatilmasa, sanash satrni ko'chirish belgisigacha davom ettiriladi. Satrni kiritishga misol:

```
string str;
cout << "str=";
getline(cin, str, '\n'); // Satrni olamiz
cout << str << endl; // Satrni chiqaramiz
```

C-satrlar bilan ishlash funksiyalari. C-satrlar bilan ishlash uchun mo'ljallangan asosiy funksiyalar bilan tanishib chiqamiz:

strlen() funksiyasi C-satrdagi belgilar sonini qaytaradi (nol belgi e'tiborga olinmaydi). Funksiyaning prototipi:

```
#include <cstring> size_t strlenfconst char *Str);
```

size_t tipi ishorasiz butun son sifatida aniqlangan. Masalan:

```
char str[7] = "String";
cout << strlen(str) << endl; // 6
```

Massivning umumiy o'lchamini sizeof operatori yordamida ham aniqlash mumkin. Operator o'lchamni baytlarda qaytaradi. char tipi '1' bayt band qilgani uchun baytdagi o'lcham massivning o'lchami bo'ladi.

```
char str[20] = "String";
cout << sizeof str << endl; // 20
```

strcpy() funksiyasi belgilarni 'S' C-satridan 'D' C-satriga nusxalaydi va nol belgi qo'yadi. Funksiyaning qiymati sifatida 'D' satriga ko'rsatkich qaytaradi. Agarda 'S' satri 'D', satridan

uzun bo'lsa, u holda buferning to'lishi yuzaga keladi. Funksiyaning prototipi:

```
#include <cstring>
char *strcpy(char *D, const char *S);
```

Funksiyadan foydalanishga misol:

```
char str[7];
strcpy(str, "String");
cout << str << endl; // String
```

VC++ da strcpy() funksiyasidan foydalanilganida ogohlantiruvchi xabar ("warning C4996") chiqariladi, shuning uchun uning o'rniga strcpy_s() funksiyasidan foydalanish lozim. Funksiyaning prototipi:

```
#include <cstring>
errno_t strcpy_s(char *D, rsize_t SizeInBytes, const char *S);
```

strcpy_s() funksiyasi 'S' satridan belgilarni 'D' satriga nusxalaydi va nol belgi qo'yadi. SizeInBytes parametrida 'D' massiv elementlarining maksimal soni ko'rsatiladi. Misol:

```
const short SIZE = 7; char str[SIZE];
strcpy_s(str, SIZE, "String");
cout << str << endl; // String
```

strncpy() funksiyasi 'S' C-satridan birinchi 'C' belgilarni 'D' C-satriga nusxalaydi. Funksiya qiymat sifatida 'D' satriga ko'rsatkich qaytaradi. Agarda 'S' satri 'D' satridan uzun bo'lsa, buferning to'lishi yuzaga keladi. Funksiyaning prototipi:

```
#include <cstring>
char *strncpy(char *D, const char *S, size_t C);
```

Agar 'S' satridagi belgilar soni 'C' sonidan kichik bo'lsa, u holda 'D' satri nol belgilar bilan to'ldiriladi. Belgilar soni ko'p bo'lsa faqat 'C'tasi nusxalanadi, bunda nol belgi avtomatik ravishda qo'yilmaydi. Oltita belgini nusxalashga misol:

```
char str[7];
strncpy(str, "String", 6);
```

```
str[6]='\'; // Nol belgi avtomatik qo'yilmaydi
cout << str << endl; // String
```

strncpy() funksiyasidan foydalanilganida ogohlantiruvchi xabar ("warning C4996") chiqariladi, shu sababli uning o'rniga strncpy_s() funksiyasidan foydalanish lozim. Funksiyaning prototipi quyidagicha:

```
#include <cstring>
errno_t strncpy_s(char *Dst, rsize_t SizelnBytes,
const char *Src, rsize t MaxCount);
```

strncpy_s() funksiyasi Src satrdan birinchi MaxCount belgini Dst satrga nusxalaydi va nolinchi belgini qo'yadi. SizelnBytes parametrida Dst massivi elementlarining maksimal soni ko'rsatiladi. Misol:

```
const short SIZE = 7; char str[SIZE];
strncpy_s(str, SIZE, "String", 5);
cout << str << endl; // Strin
```

strcat() funksiyasi 'S' C-satrdan belgilarni 'D' C-satr oxiriga nusxalaydi va nol belgi qo'yadi. Qiymat sifatida 'D' satrga ko'rsatkichni qaytaradi. 'D' satrining o'lchami kamlik qilsam buferning to'lishi amalga oshqadi. Funksiyaning prototipi:

```
#include <cstring>
char *strcat(char *D, const char *S);
```

Funksiyadan foydalanishga misol:

```
char str[20] = "Bir";
strcat(str, "Ikki");
strcat(str, "Uch");
cout << str << endl; // BirIkkiUch
```

Nusxalashdan oldin 'D' satrda nolinchi belgi bo'lishi lozim. Lokal o'zgaruvchilar avtomatyik nomlanmaydi, shu sababli kod xatolikka olib keladi. Xatolikdan holi bo'lish uchun satrni nol bilan nomlash mumkin.

```
char str[20]; // Lokal o'zgaruvchi
strcat(str, "Bir"); // Xato, nol belgi yo'q
char str[20] = {0}; // Nollar bilan nomlash
```

strcat(str, "Bir"); // *Hammasi joyida*
cout << str << endl; // *Bir*

VC++da strcat() funksiyasidan foydalanilganda ("warning C4996") ogohlantiruvchi xabar chiqariladi, shu sababli uning o'rniga strcat_s() funksiyasidan foydalanish tavsiya qilinadi. Funksiyaning prototipi:

#include <cstring>
errno_t strcat_s(char *D, rsize_t SizeInBytes, const char *S);

strcat_s() funksiyasi belgilarni 'S' satridan 'D' satri oxiriga nusxalaydi. SizeInBytes parametrida 'D' massiv elementlarining maksimal soni ko'rsatiladi. Misol:

const short SIZE = 20;
char str[SIZE] = {0};
strcat_s (str, SIZE, "Bir");
strcat_s(str, SIZE, "Ikki");
strcat_s(str, SIZE, "Uch");
cout << str << endl; // *BirIkkiUch*

strncat() funksiyasi Source C-satridan birinchi Count belgini Dest satrining oxiriga nusxalaydi va nolinchi belgini qo'yadi. Funksiyaning qiymati sifatida Dest satrga ko'rsatkich qaytaradi. Nusxalashdan oldin satrda albatta nol belgi bo'lishi lozim. Dest satr yetarlicha o'lchamga ega bo'lmasa, buferning to'lishi yuzaga keladi.

Funksiyaning prototipi:

#include <cstring>
char *strncat(char *Dest, const char *Source, size_t Count);

Funksiyadan foydalanishga misol:

char str[20] = "Bir";
strncat(str, "Ikki", 3);
strncat(str, "Uch", 3);
cout << str << endl; // *BirIkkUch*

VC++da strncat() funksiyasidan foydalanilganda ogohlantiruvchi xabar ("warning C4996") chiqariladi, shu sababli strn-

cat() o'rniga strncat_s() funksiyasidan foydalanish lozim. Funksiyaning prototipi:

```
#include <cstring>
errno_t strncat_s(char *Dst, rsize_t SizeInBytes,
const char *Src, rsize_t MaxCount);
```

strncat_s() funksiyasi birinchi MaxCount belgini Src satridan Dst satri oxiriga nusxalaydi. SizeInBytes parametrida Dst massivi elementlarining maksimal soni ko'rsatiladi. Misol:

```
const short SIZE = 20;
char str[SIZE] = "Bir";
strncat_s(str, SIZE, "Ikki", 3);
strncat_s(str, SIZE, "Uch", 3);
cout << str << endl; // BirIkkUch
```

strtok() funksiyasi str C-satrni, Delim C-satr beglilaridan ajratkich sifatida foydalangan holda, satr ostlariga ajratadi. Birinchi chaqiruvda ikkala parametr ko'rsatiladi. Quymat sifatida birinchi satrostiga ko'rsatkich yoki str C-satrda belgi-ajratkichlar topilmasa, nolinchi ko'rsatkich qaytariladi. Navbatdagi satrostilarni olish uchun birinchi parametrda nolinchi ko'rsatkichni ikkinchi parametrda avvalgi belgi-ajratkichlarni ko'rsatgan holda strtok() funksiyasini chaqirish lozim. Funksiyaning prototipi:

```
#include <cstring>
char *strtok(char *Str, const char *Delim);
```

Funksiyadan foydalanishga misol:

```
char str[] = "a,b.c=d", *p = 0;
p = strtok(str, ",.=");
while (p) {
 cout << p << "-";
 p = strtok(0, ",.=");
} // Natija: a-b-c-d-
cout << endl;
```

strtok() funksiyasidan foydalanishda ogohlantiruvchi xabar chiqariladi (*"warning C4996"* – bufer to'lib qolishi mumkin),

shu sababli uning o'rniga strtok_s() funksiyasidan foydalanish tavsiya etiladi. Funksiyaning prototipi:

```
#include <cstring>
char *strtok_s(char *Str, const char *Delim, char **Context);
```

Birinchi ikki parametr strtok() funksiyasi parametrlari bilan bir xil. Context parametrida ko'rsatkich manzili uzatiladi. Misol:

```
char str[] = "a b c,d", *p = 0, *context = 0;
p = strtok_s(str, ",", &context) ;
while(p) {
 cout << p << "-";
 p = strtok_s(0, " ," &context);
cout << endl;
} // Natija: a-b-c-d-
```

Deyarli barcha standart funksiyalar "warning C4996" ogohlantiruvchi xabarini chiqaradi. Chunki funksiyalar ko'rsatilmaganda berilganlarni korrektlikka tekshirmaydi. Bunday holda bufer to'lib qolishi mumkin. O'z harakatlaringizga ishonchingiz komil bo'lsa, ogohlantiruvchi xabarlarni chiqarishni "bekitib qo'yish" mumkin. Bunda berilganlarning korrektligi (to'g'riligi) uchun butun mas'uliyat dasturchining zimmasiga yuklatiladi. Bekitib qo'yishni ikki usul bilan amalga oshirish mumkin.

1-usul. Sarlavha fayllarini ulashdan avval dasturning boshida _CRT_SECURE_NO_WARNINGS nomli makrosni aniqlab olish. Misol:

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
```

2-usul. warning pragmasini qo'yish. Misol:

```
#pragma warning(disable: 4996)
```

#define va #pragma direktivalari oxirida nuqtali vergul qo'yilmaydi.

Xabar chiqarishni o'chirib qo'yish o'rniga nomlari "_s" (masalan, strcpy() o'rniga strcpy_s()) bilan tugaydigan funksiyalardan yoki ular avtomatik "_s"-analoglarga almashishi uchun funksiyani qayta yuklashdan foydalanish mumkin. Buning uchun dasturning boshida sarlavha fayllarini ulashdan avval 1 qiymatli _crt_secure_cpp_overload_ standard_names makrosni aniqlash lozim. Bu makros aniq belgilar soni ko'rsatiluvchi funksiyalarni (masalan, strncpy() funksiyasi) almashtirishni bajarmaydi. Bu funksiyalarni qayta yuklash uchun qo'shimcha ravishda qiymati 1ga teng bo'lgan _crt_secure_cpp_overload_ standard_names_count makrosni aniqlab olish lozim.

Quyida satrli funksiyalarni qayta yuklash listinggi keltirilgan.

```
#define _CRT_SECURE_CPP_OVERLOAD_STAN-
DARD_NAMES 1
#define _CRT_SECURE_CPP_OVERLOAD_STAN-
DARD_NAMES_COUNT 1
#include <iostream>
#include <cstring>
int main() {
 char str1[7] = {0}, str2[7] = {0};
 strcpy(str1, "String"); cout << str1 << endl; // String
 strncpy(str2, "String", 5); cout << str2 << endl; // Strin
 cin.get();
 return 0;
}
```

```
String
Strin
```

C-satr belgilariga kirish. C-satr aniqlanganidan keyin o'zgaruvchida birinchi belgining manzili saqlanadi. Ya'ni, o'zgaruvchining nomi satrning birinchi belgisiga murojaat qiluvchi ko'rsatkich hisoblanadi. Shu sababli, massiv elementiga kirish kvadrat qavs ichida ko'rsatilgan indeks bo'yicha (indeks noldan boshlanadi) yoki manzilli arifmetikadan foydalanilgan

holda amalga oshirilishi mumkin. Masalan, quyidagi chiqarish ko'rsatmalari ekvivalent:

```
char str[] = "String";
cout << str[1] << endl; // t
cout << *(str+1) << endl; // t
```

Bunday yo'l bilan belgini nafaqat olish, balki o'zgartirish ham mumkin:

```
char str[] = "String";
str[0]='s'; // Indeks yordamida o'zgartirish
*(str + 1)='T'; // Ko'rsatkich yordamida o'zgartirish
cout << str << endl; // sTring
```

Alohida belgi qo'shtirnoq ichida emas, balki apostroflar ichida ko'rsatilishini e'tiborga olish lozim. Agar belgi qo'shtirnoq ichida ko'rsatilsa, bir belgi o'rniga ikki belgi bo'ladi: belgining o'zi va hol belgi.

Quyidagicha yo'l bilan ko'rsatkichni e'lon qilish va unga satr manzilini o'zlashtirish mumkin:

```
char *p = 0;
char str[] = "String";
p = str;
+p = 's';
++p; // Ko'rsatkichni ikkinchi belgiga ko'chirish
p = 'T';
cout << str << endl; // sTring
```

E'tibor bering, satr nomidan oldin & operatori ko'rsatilmadi, chunki o'zgaruvchining nomi birinchi belgining manzilini oladi. agar operatordan foydalanilsa, u holda qo'shimcha ravishda kvadrat qavs ichida indeksni ko'rsagtish lozim.

```
p = &str[0]; // p = str; ga ekvivalent
```

Ko'rsatkich nomlashda unga satr o'zlashtirish mumkin. Bunday satrlarni o'zgartirish tavsiya etilmaydi, shu sababli, odatda tipdan oldin const kalit so'zi ko'rsatiladi. Misol:

```
const char *str = "String";
```

```
cout << str << endl; // String
```

C-***satr belgilarini saralash*** uchun for siklidan foydalanish qulay. Siklning brinchi parametrida o'zgaruvchi-hisoblagichga nol ('0') qiymat o'zlashtiriladi (satrda belgilar noldan boshlab tartiblanadi), satrdagi belgilar sonidan kam qiymat o'zgaruvchi-hisoblagichning davom ettirish sharti hisoblanadi. Satrdagi belgilar sonini strlen() funksiyasi orqali olish mumkin. Funksiyaning prototipi:

```
#include <cstring>
size_t strlen(const char *Str);
```

strlen() funksiyasi nol belgini hisobga olmagan holda belgilar sonini qaytaradi. size_t tipi ishorasiz butun sifatida aniqlangan. for siklining uchinchi parametrida siklning har bir takrorlanishida (iteratsiya) bittaga orttirish ko'rsatiladi. Satrning barcha belgilarini to'g'ri va teskari tartibda chiqarish dasturi kodini yozamiz:

```
char str[] = "String";
int len = strlen(str);
// Belgilarni to'g'ri tartibda chiqarish
for(int i = 0; i < len; i++) {
 cout << str[i] << endl;
}
cout << " " << endl;
// Belgilarni teskari tartibda chiqarish
for(int j = len - 1; j >= 0; j--) {
 cout << str[j] << endl;
}
```

Bu misolda belgilar soni len o'zgaruvchisida sikldan tashqarida saqlanadi. Agar strlen() funksiyasi shart ichida ko''rsatilsa, u holda belgilar soni siklning har bir iteratsiyasida hisoblanadi. shu sababli, uni sikldan tashqarida olgan yoki for siklining birinchi parametrida o'zlashtirgan ma'qul.

```
for(int i = 0, len = strlen(str); i < len; i++) {
 cout << str[i] << endl;
```

C-satr belgilarini ko'rsatkich va for sikli yordamida chiqatish quyidagicha bajariladi:

```
char str[] = "String";
for(char *p = str; *p; p++) {
 cout << *p << endl;
}
```

Bunda boshlang'ich qiymat birinchi belgining manzili hisoblanadi. Siklning davom etish sharti korsatkich murojaat qilayotgan qiymat hisoblanadi. '0' koda ega bo'lgan nol belgidan tashqari, har qanday belgi true sifatida qaraladi. C-satr nol belgi vbilan tugallanganligi sababli, bu belgi false qiymat qaytaradi va sikl tugallanadi. for siklining uchinchi parametrida har bir iteratsiyada ko'rsatkichning birga ortishi ko'rsatiladi. Bu holda *manzilli arifmetika* qoidasidan foydalanilladi. har doim for sikli o'rniga while siklidan foydalanish mumkin:

```
char *p = 0;
char str[] = "String";
p = str;
while (*p) {
 cout << *p++ << endl;
}
p = str; // Ko'rsatkichning holatini tiklash
```

Keltirib o'tilgan misolni tahlil qilib chiqamiz. Boshlanishda ko'rsatkich va satr e'lon qilinadi. Keyin ko'rsatkichga birinchi belgining manzili o'zlashtiriladi. Ko'rsatkich murojaat qilayotgan qiymat nol belgiga teng bo'lmaguncha while sikli bajariladi. while sikli ichida ko'rsatkich murojaat qilayotgan belgi chiqariladi, so'ngra ko'rsatkich navbatdagi belgiga ko'chadi (*p++). p++ ifoda joriy manzilni qaytaradi, so'ngra uni bir birlikka orttiradi. '*' belgisi ko'rsatilgan manzil bo'yicha belgini olish imkonini beradi. Bajarilish ketma-ketligi qavsning quyidagicha *(p++) qo'yilishiga yoki ikkita ko'rsatma: *p va p=p+1 ga mos keladi.

C-*satrda qidirish va almashtirish* uchun foydalanilanish mumkin bo'lgan funksiyalar bilan tanishamiz.

strchr() funksiyasi str C-satrida "ch" belgining birinchi uchrashini qidiradi. Qiymat sifatida str C-satrida birinchi topilgan belgiga ko'rsatkich yoki agar belgi topilmasa, nolinchi ko'rsatkich qaytaradi. Funksiyaning prototiplari quyidagicha ko'rinishga ega:

```
#include <cstring>
char *strchr(char *Str, int Ch) ;
const char *strchr(const char *Str, int Ch) ;
```

Funksiyadan foydalanishga misol:

```
char str[] = "string string", *p = 0;
p = strchr(str, 'n');
if(p) {
 cout << "Index: " << p – str << endl;
} // Natija: Index: 4
```

strrchr() – str C-satrida "ch" belgining oxirgi uchrashini qidiradi. Qiymat sifatida belgiga ko'rsatkich, agar belgi topilmasa, nolinchi ko'rsatkich qaytaradi.

Funksiyaning prototiplari quyidagicha:

```
#include <cstring>
char *strrchr(char *Str, int Ch) ;
const char *strrchr(const char *Str, int Ch) ;
```

Misol:

```
char str[] = "string string",*p = 0;
p = strrchr(str, 'n');
if(p) {
 cout << "Index: " << p – str << endl;
}
```

```
Index: 11
```

memchr() funksiyasi St satrda K belgining birinchi uchrashini qidiradi. Funksiya qiymat sifatida St satrda birinchi topilgan belgiga ko'rsatkichni qaytaradi. Agar belgi topilmasa, nolinchi

ko'rsatkich qaytaradi. Natijani ko'rsatkichga o'zlashtirishdan avval char* tipiga keltirishni bajarish zarur. Funksiyaning prototipi:

```
#include <cstring>
void *memchr(void *St, int K, size_t N);
```

Misol tariqasida satrda "s" belgini qidish kodini ko'ramiz. Belgining indeksini chiqarish uchun manzilli arifmetikadan foydalanamiz (bir ko'rsatkichni boshqasidan ayirish):

```
char str[]="String", *p = 0;
p = (char *)memchr(str, 's', strlen(str));
if(p) {
 cout << "Index: " << p – str << endl;
} else cout << "Topilmadi" << endl;
```

```
Topilmadi
```

strpbrk() – str C-satrida Control C-satriga kiruvchi belgilardan birining birinchi uchrashini qidiradi (nol belgi hisobga olinmaydi). Funksiya qiymat sifatida birinchi topilgan belgiga ko'rsatkich yoki belgilar topilmasa, nolinchi belgiga ko'rsatkich qaytaradi. Funksiyaning prototiplari:

```
#include <cstring>
char *strpbrk(char *Str, const char *Control);
const char *strpbrk(const char *Str, const char *Control);
```

Misol:

```
char str[] = "string string", *p = 0;
p = strpbrk(str, "nr");
if(p) {
 cout << "Index: " << p - str << endl;
} // Natrija: Index: 2 (birinchi "r" harfining indeksi)
```

strcspn() funksiyasi C-satriga kiruvchi Control belgilardan biri bilan mos keluvchi str C-satridagi birinchi belgi indeksini qaytaradi.

Funksiyaning prototipi:

```
#include <cstring>
```

```
size_t strcspn(const char *Str, const char *Control);
Misol:
char str[] = "string1 string2";
size_t index;
index = strcspn(str, "ingr");
cout << "Index: " << index << endl; // Natija: Index: 2 ("r" harfi)
```

strspn() funksiyasi C-satriga kuruvchi Control belgilarning hech biri bilan mos kelmaydigan str C-satridagi birinchi belgi indeksini qaytaradi. Funksiyaning prototipi:

```
#include <cstr ing>
size_t strspn(const char *Str, const char *Control);
Misol:
char str[] = "string1 string2";
size_t index;
index = strspn(str, "singtr");
cout << "Index: " << index << endl;
// Natija: Index: 6 ("1" "singtr"ga kirmaydi)
```

strstr() – str C-satrida substr C-satridagi butun parchaning birinchi uchrashini qidiradi. Qiymat sifatida parchaning birinchi uchrashiga ko'rsatkich yoki uchramasa, nol ko'rsatkich qaytaradi. Funksiyaning prototiplari:

```
#include <cstring>
char *strstr(char *Str, const char *SubStr);
const char *strstr(const char *Str, const char *SubStr);
Misol:
char str[] = "string1 string2", *p = 0;
p = strstr(str, "ing2");
if(p) {
 cout << "Index: " << p – str << endl;
} // Natija: Index: 11
```

memset() funksiyasi Dst satrdagi birinchi Size belgilarni Val belgisi bilan almashtiradi. Funksiya qiymat sifatida

Dst satrga ko'rsatkichni qaytaradi. Quymatni ko'rsatkichga o'zlashtirishdan avval char* tipga keltirishni bajarish lozim. Funksiyaning prototipi quyidagicha:

```
#include <cstring>
void *memset(void *Dst, int Val, size_t Size);
```

Misol:

```
char str[] = "String", *p = 0;
p = (char *)memset(str, '*', 4);
if(!p) exit(1);
cout << str << endl;
cout << p << endl;
```

C-satrlarni taqqoslash uchun quyidagi funksiyalardan foydalaniladi:

strcmp() funksiyasi Str1 C-satrni Str2 C-satr bilan taqqoslaydi va quyidagi qiymatlardan birini qaytaradi:

- Str1 < Str2 bo'lsa, manfiy son;
- Str1 = Str2 bo'lsa, 0 qiymat;
- Str1 > Str2 bo'lsa, musbat son.

Taqqoslash belgilar registrni hisobga olgan holda amalga oshiriladi. Funksiyaning prototipi quyidagicha:

```
#include <cstring>
int strcmp(const char *Str1, const char *Str2);
```

Misol:

```
char str1[] = "абв", str2[] = "абв";
int x = 0;
x = strcmp(str1, str2);
cout << x << endl; // 0
str1[2] = 'б'; // str1[] = "абб", str2[] = "абв";
x = strcmp(str1, str2);
cout << x << endl; // -1
str1[2] = 'г'; // str1[] = "абг", str2[] = "абв";
x = strcmp(str1, str2);
```

cout << x << endl; // *1*

strncmp() funksiyasi Str1 va Str2 C-satrlardagi birinchi Count belgilarni taqqoslasydi. Agar nolinchi bayt avval uchrasa, Count parametrining qiymati e'tiborga olinmaydi. Funksiya quyidagi qiymatlardan birini qaytaradi:

- Str1 < Str2 bo'lsa, manfiy son;
- Str1 = Str2 bo'lsa, 0 qiymat;
- Str1 > Str2 bo'lsa, musbat son.

Taqqoslash belgilar registrni hisobga olgan holda amalga oshiriladi. Funksiyaning prototipi:

```
#include <cstring>
int strncmp(const char *Str1, const char *Str2, size_t Count);
```

Misol:

```
char str1[]="абв", str2[]="абг";
int x = 0;
x=strncmp(str1, str2, 2);
cout << x << endl; // 0
x=strncmp(str1, str2, 3);
cout << x << endl; // -1
```

strcoll() funksiyasi strcmp() funksiyasi bilan bir xilda, ammo taqqoslash joriy lokalda LC_COLLATE qiymatni hisobga olgan holda amalga oshiriladi. Masalan, "ё" harfi "е" va "ж" oraliqqa tushmaydi chunki, cp1251 kodlashida "ё" harfi 184 kodiga ega, "е" – 229, "ж" harfi esa – 230. Agar strcmp() funksiyasi yordamida taqqoslash amalga oshirilsa, "е" harfi "ё" harfidan katta bo'ladi (229 > 184). strcoll() funksiyasidan foydalanilganda lokalning "Russian_Russia.1251" sozlanishida "ё" harfi "е" va "ж" harflari oralig'iga tushadi, chukni lokal alifbo bo'yicha sozlanadi. Lokal sozlanmagan bo'lsa strcmp() funksiyasi bilan bir xilda bo'ladi. Taqqoslash registrni hisobga olgan holda bajariladi. Funksiyaning prototipi:

```
#include <cstring>
int strcoll(const char *Str1, const char *Str2);
```

Misol ko'raylik.

```
setlocale(LC_COLLATE, , "Russian_Russia.1251");
char str1[] = "e", str2[] = "ë";
int x = strcoll(str1, str2);
cout << x << endl;
// 1 (lokalni sozlamasdan: e (kodi 229) > ë (kodi 184))
// -1 (lokal sozlangandan keyin: e < ë)
```

strxfrm() funksiyasi Src C-satrini maxsus formatdagi satrga almashtiradi va uni Dst ga yozadi. Oxiriga nol-belgi qo'yiladi MaxCount dan ortmagan berlgi yozilsdi. Kerakli belgilarni olish (nol belgisiz) uchun MaxCount parametrida 0 soni ko'rsatiladi, Dst parametrida nol belgi beriladi. Satrlarni taqqoslash ko'p marotaba amalga oshirilayotgan bo'lsa, strxfrm() funksiyasidan foydalanish tavsiya qilinadi. Funksiyaning prototipi:

```
#include <cstring>
size_t strxfrm(char *Dst, const char *Src, size_t MaxCount);
Misol:
setlocale(LC_ALL, "Russian_Russia.1251") ;
int x = 0;
char str1[] = "e", str2[] = "ë";
char buf1[10] = {0}, buf2[10] = {0};
x = strxfrm(0, str1, 0);
cout << x << endl; // 6 (bufer: 6+1)
strxfrm(buf1, str1, 10);
strxfrm(buf2 , str2, 10);
x = strcmp(str1, str2);
cout << x << endl; // 1 (e > ë)
x = strcmp(buf1, buf2);
cout << x << endl; // -1 (e < ë)
```

_stricmp() funksiyasi C-satrlarni registrni hisobga olmagan holda taqqoslaydi. Rus harflari uchun lokalni sozlash zarur bo'ladi. Bu funksiya faqat VC++ da mavjud va standartga kirmaydi. Funksiyaning prototipi:

```
#include <cstr ing>
int stricmp(const char *Str1, const char *Str2);
```

Misol:

```
setlocale(LC_CTYPE, "Russian_Russia.12 51");
char s1[] = "абв", s2[] = "АББ";
cout << _stricmp(s1, s2) << endl; // 0
```

Satrlarni taqqoslash uchun memcmp() funksiyasidan ham foydalanish mumkin. memcmp() funksiyasi Mas1 va Mas2 massivlarning birinchi Size baytlarini taqqoslaydi. Qiymat sifatida quyidagilardan birini qaytaradi:

- manfiy son – agar Mas1 < Mas2;
- 0 – agar Mas1 = Mas2;
- musbat son – agar Mas > Mas2.

Funksiyaning prototipi:

```
#include <cstring>
int memcmp(const void *Bufl, const void *Buf2, size_t Size);
```

Misol:

```
char str1[] = "abc", str2[] = "abc"; int x = 0;
x = memcmp(str1, str2, sizeof str2);
cout << x << endl; // 0
str1[2] = 'b';
x = memcmp(str1, str2, sizeof str2);
cout << x << endl;
str1[2] = 'd';
x = memcmp(str1, str2, sizeof str2);
cout << x << endl;
```

memcmp() funksiyasi belgilarning registrni hisobga olgan holda taqqoslashni amalga oshiradi. Belgilar registrni hisobga olgmagan holda taqqoslash zarur bo'lsa, _memicmp() funksiyasidan foydalanish mumkin. Funksiyaning prototipi:

```
#nclude <cstring>
```

int _memicmp(const void *Buf1, const void *Buf2, size_t Size);

Funksiyadan foydalanishga misol:

```
std::setlocale(LC_ALL, "Russian_Russia.1251");
char str1[] = "абв", str2[] = "АБВ"; int x = 0;
x = memicmp(str1, str2, sizeof str2);
cout << x << endl; // 0
x = memcmp(str1, str2, sizeof str2);
cout << x << endl; // 1
```

Registrni e'tiborga olmagan holda belgilarni taqqoslash uchun _stricmp() va _memicmp() funksiyalaridan foydalanish mumkin. Bu funksiyalar standartga kirmaganligi sababli, boshqa kompilyatorlarda bajarilmasligi mumkin. Shu sababli, registrni e'tiborga olmagan holda belgilarni taqqoslash uchun, o'z funksiyamizni (casecmp()) yozamiz.

```
#include <iostream>
#include <clocale>
#include <cstring>
#include <cctype>
int casecmp(const char *str1, const char *str2);
int main() {
 char s1[] = "абв", s2[] = "АБВ";
 cout << strcmp(s1, s2) << endl; // 1
 cout << casecmp(s1, s2) << endl; // 0
 cout << casecmp("аб", "аб") << endl; // 0
 cout << casecmp(s1, "АБ") << endl; // 226
 cout << casecmp("аба", s2) << endl; // -2
 cin.get();
 return 0;
}
int casecmp(const char *str1, const char *str2) {
 setlocale(LC_CTYPE, "Russian_Russia.1251");
 int ch1 = 0, ch2 = 0, result = 0;
```

```
while(1) {
ch1 = tolower((unsigned char)*str1);
ch2 = tolower((unsigned char)*str2);
result = ch1 – ch2;
// Agar qiymatlar teng bo'lmasa, natijani qaytaramiz
if(result) return result;
// Satr oxiriga borsa, u holda satrlar teng
if(!ch1 && !ch2) return 0;
++str1; // Ko'rsatkichlarni ko'chiramiz
++str2;
}
return 0;
}
```

casecmp() funksiyasi parametrlar sifatida ikkita satrning (str1 va str2) manzilini oladi va satrlar teng bo'lsa, '0' qiymat qaytaradi. Agar str1 < str2 bo'lsa manfiy va str1 > str2 bo'lsa musbat son qaytaradi. Parametr sifatida satrli o'zgarmasni uzatish mumkin bo'lishi uchun parametrlardan oldin const (masalan, const char *str1) kalit so'zi ko'rsatiladi. Sikl ichida tolower() funksiyasi yordamida ikkala belgi bir registrga keltiriladi, so'ngra ikkinchi satr belgisidan birinchi satr belgisi ayiriladi. Ayirish natijasi nolga teng bo'lmasa, bu qiymat funksiyaga qaytariladi. Nolga teng bo'lsa, ikkita satr oxiriga borganligi tekshiriladi. Bunda satrlar teng bo'lsa 0 qiymatni qaytarish mumkin. Satr oxiriga borilmagan bo'lsa, ko'rsatkich navbatdagi belgiga ko'chiriladi va taqqoslash amalga oshiriladi.

9.2. String sinfi

C-satrlar massiv hisoblanadi, shu sababli bunday satrlar bilan ishlash qulay emas. Masalan, satr e'lon qilinganidan keyin qo'shtirnoq ichida qiymat o'zlashtirish mumkin emas:

```
char str1[] = "abc"; // Normal
char str2[7];
str2 = "abc"; // Xato
```

Bundan tashqari, e'lon qilinganidan so'ng satr o'lchamini o'zgartirish yoki operatorlar yordamida ikkita satrni birlashtirish mumkin emas:

```
char str1[] = "abc", str2[7] = "def";
char str3[] = str1 + str2; // Xato
```

string sinfi dasturchining belgili massiv o'lchamini nazorat qilib borish zaruratidan halos qiluvchi qulay interfeysni taqdim etadi va satrga ishlov berish uchun ko'plab usullarni taqdim etadi. string sinfi obyektini C-satrga almashtirish mumkin. string sinfining nomi basic_string shablon sinfining soxtasi (*psevdonim*) hisoblanadi halos:

```
typedef basic_string<char, char_traits<char>,
allocator<char> > string;
```

string sinfidan foydalanish uchun string faylini ulash zarur:

```
#include <string>
```

string sinfi obyektlarini ">>" va "<<" operatorlari yordamida mos ravishda kiritish va chiqarish mumkin. Bunda shuni hisobga olish lozimki, ">>" operatori yordamida birinchi bo'shliq belgisigacha bo'lgan parchani chiqarish mumkin. Qiymatlarni kiritish va chiqarishga doir misollar:

```
string str;
cout << "str = ";
cin >> str; // Birinchi bo'shliqqacha satrni olamiz
cout << str << endl; // Satrni chiqarish
```

Satrni e'lon qilish va nomlash. string sinfi obyektini e'lon qilish uchun quyidagi usullardan biridan foydalanish mumkin:

a) sinf namunasini nomlamagan holda e'lon qilish. Buning uchun o'zgaruvchi nomidan oldin sinf nomi ko'rsatiladi. Satr uzunligini ko'rsatilmaydi. Masalan: string str;

b) C-satrni qavs ichida yoki "=" operatoridan keyin (obyektni nomlash amalga oshirladi) ko'rsatish. Masalan:

string str1("String1");

string str2 = "String2";

Agar ikkinchi parametrda son berilsa, satrning bir qismini o'zlashtirish mumkin (hammasini emas).

string str("String", 3); *// Str*

c) string sinfi obyektini qavs ichida yoki "=" operatoridan keyin ko'rsatish:

string str1("String");

string str2(str1); yoki string str3 = str1;

Obyektning hammasini emas, uning bir qismini o'zlashtirish mumkin. Buning uchun ikkinchi parametrda boshlang'ich indeksni, uchinchi parametrda esa oxirgi indeksni uzatish lozim. Oxirgi indeks ko'rsatilmasa, u holda parcha satr oxirigacha nusxalanadi. Misol:

string str1("String");

string str2(str1,3); *// ing*

string str3(str1,0,3); *// Str*

d) birinchi parametrda belgilar sonini, ikkinchi parametrda belgi-to'ldirgichni ko'rsatish:

string str(5,'s'); *// sssss*

O'zgaruvchini e'lon qilgandan keyin "=" operatori yordamida qiymat o'zlashtirish mumkin. Satrning uzunligini nazorat qilib borish shart emas. Agar string sinfidagi obyekt o'zlashtirilayotgan bo'lsa, unga murojaat o'zlashtirilmaydi, balki boshlang'ich obyektning nusxasi yaratiladi:

strmg str1, str2;

str1 = "string"; *// C-satrni o'zlashtirish*

str1 = "new string"; *// Satrning uzunligini nazaorat qilish shart emas*

str2 = str1; *// Obyektga string sinfini o'zlashtirish*

Qiymatlarni o'zlashtirish uchun "=" operatori o'rniga assign() metodidan foydalanish mumkin. Metodning nomi o'zgaruvchi nomidan keyin nuqta bilan ko'rsatiladi. Metodning prototiplari:

```
string &assign(size_type Count, char Ch);
string &assign(const char *Ptr);
string &assign(const char *Ptr, size_type Count);
string &assign(const string &Right);
string &assign(const string &Right, size_type Roff,
size_type Count);
templateCclass It> string &assign(_It First, It Last);
string &assign(const_pointer First, const_pointer Last);
string &assign(const_iterator First, const_iterator Last);
```

Birinchi prototip Ch belgini Count marta o'zlashtiradi. Ikkinchi prototip C-satrni to'laligicha o'zlashtirish imkonini beradi. Uchinchi prototip – Ptr C-satrdan faqat birinchi Count belgilarni, to'rtinchi prototip string sinfidagi obyektni to'laligicha o'zlashtiradi, beshinchi prototip esa – Roff indeksidan boshlab, faqat Count ta belgini. Qolgan prototiplar iteratorlar yordamida boshlang'ich (First) va oxirgi (Last) pozitsiyalarni ko'rsatish imkonini beradi. Misol keltiramiz:

```
#include "stdafx.h"
#include <iostream>
#include <string>
using namespace std;
using namespace System;
int main() {
 string str1, str2("BC");
 char str3[] = "DEF";
 str1.assign(2,'+'); // 1-Prototip
 cout << str1 << endl; //++
 str1.assign(str3); // 2-Prototip
 cout << str1 << endl; // DEF
 str1.assign(str3, 2); // 3-Prototip
 cout << str1 << endl; // DE
 str1.assign(str2); // 4-Prototip
 cout << str1 << endl; // BC
```

```
    str1.assign(str2, 0, 1); // 5-Prototip
    cout << str1 << endl; // B
    str1.assign(str2.begin(), str2.end()); // 6-Prototip
    cout << str1 << endl; // BC
    cin.get();
    return 0;
}
```

```
++
DEF
DE
BC
B
BC
```

Obyektini C-satr yoki belgilar massiviga almashtirish. string sinfi obyektini boshqa tipga o'zgartirish uchun quyidagi metodlar tayinlangan:

c_str() – const char* tipidagi ko'rsatkich qaytaradi. Undan C-satrni string sinfi obyekti o'rniga uzatish (masalan funksiyaga) zarur bo'lgan vaziyatlarda foydalaniladi. Bunday satrni o'zgartirish mumkin emas. Obyekt o'zgartirilgandan keyin ko'rsatkich noto'g'ri bo'lib qolishi mumkinligini hisobga olish lozim. Shu sababli, obyektni o'zgartirgandan keyin ko'rsatkichning qiymatini yangilash lozim. Metodning prototipi:

```
#include <string>
const char *c_str() const;
```

string sinfi obyektini C-satrga almashtirishga doir misol:

```
string str1("String");
char str2[80] ;
strcpy_s(str2, 80, str1.c_str());
cout << str2 << endl; // String
```

data() – const char* tipidagi ko'rsatkich qaytaradi. data() metodi c_str() metodidan farqli ravisha C-satrga emas, balki belgilar massiviga ko'rsatkich qaytaradi (nolinchi belgi qo'shilmaydi). Nolinchi belgi bilan tugallanuvchi satrni olish zarur bo'lsa, c_str() metodidan foydalaniladi. Obyekt o'zgartirilgandan keyin ko'rsatkich noto'g'ri bo'lib qolishi mumkinligini hisob-

ga olish lozim. Shu sababli, obyektni o'zgartirgandan keyin ko'rsatkichning qiymatini yangilash lozim. Metodning prototipi quyidagicha:

```
#include <string>
const char *data() const;
```

Misol:

```
str ing str("String");
const char *p = 0;
p = str.dat a();
cout << p[0] << endl; // S
cout << p[5] << endl; // g
```

copy() – Ptr belgilar massiviga off indeksidan boshlab Count ta belgilarni nusxalaydi. Agar indeks ko'rsatilmasa, belgilar satr boshidan boshlab nusxalanadi. Nolinchi belgi avtomatik qo'shilmaydi. Metod qiymat sifatida nusxalangan belgilar sonini qaytaradi. Metodning prototipi:

```
#include <string>
size_type copy(char *Ptr, size_type Count, size_type off=0) const;
```

copy() metodidan foydalanishga misol:

```
string str1("String");
char str2[20] = {0}, str3[20] = {0};
basic_string<char>::size_type count;
count = str1.copy(str2, 5);
cout << count << endl; // 5
cout << str2 << endl; // Strin
str1.copy(str3, 3, 1);
cout << str3 << endl; // tri
```

VC++ dasturida bu metoddan foydalanilganda ogohlantiruvchi xabar (*"warning C4996"*) chiqariladi, shu sababli uning o'rniga _Copy_s() metodidan foydalangan ma'qul. Metodning prototipi quyidagicha:

```
#include <cstring>
```

size_type _Copy_s(char *Dest, size_type Dest_size,
size_type Count, size_type Off = 0) const;

_Copy_s() metodi Off indeksidan boshlab Count belgilarni Dest belgili massivga nusxalaydi. Agar Off indeksi ko'rsatilmasa, satr boshidan boshlab nusxalanadi. Dest_size parametrida Dest belgili massivning o'lchami ko'rsatiladi. Misol:

```
string str1("String");
char str2[20] = {0}, str3[20] = {0};
basic_string<char>::size_type count;
count = str1._Copy_s(str2, sizeof str2, 5);
cout << count << endl; // 5
cout << str2 << endl; // Strin
str1._Copy_s(str3, sizeof str3, 3, 1);
cout << str3 << endl; // tri
```

Satr o'lchamini olish va o'zgartirish. Satr o'lchamini (uzunligini) olish va o'zgartirish uchun quyidagi metodlardan foydalaniladi:

size() va length() – joriy vaqtda satrdagi belgilar sonini qaytaradi. Metodlarning prototiplari:

```
#include <string> size_type size() const;
size_type length() const;
```

Misol:

```
string str("String");
cout << str.size() << endl; // 6
cout << str.length() << endl; // 6
```

capacity() – xotirani qayta taqsimlamasdan satrga yozish mumkin bo'lgan belgilar sonini qaytaradi. Metodning prototipi:

```
#include <string> size_type capacity() const;
```

Misol:

```
string str("String");
cout << str.size() << endl; // 6
```

```
cout << str.capacity() << endl; // 15
str += "string2 string3";
cout << str.size() << endl; // 22
cout << str.capacity() << endl; // 31
```

reserve() – xotirani qayta taqsimlamasdan satrga yozish mumkin bo'lgan belgilarning minimal sonini berish uchun foydalaniladi. Avvalgi misoldan ko'rinadiki, qo'shimcha xotira ajratish ma'lum bir zahira bilan avtomatik amalga oshiriladi. Agar satrga yozish tez-tez amalga oshirilsa, xotiraning taqsimlanishi bir necha bor qayta taqsimlanganligi uchun dasturning samaradorligini pasaytirishi mumkin. Shu sababli, belgilar soni avvaldan ma'lum bo'lsa, u holda uni reserve() metodi yordamida ko'rsatish kerak. Metodning prototipi:

```
#include <string> void reserve(size_type Newcap = 0);
```

Satrning minimal o'lchamini ko'rsatishga misol:

```
string str("String");
str.reserve(50);
cout << str.size() << endl; // 6
cout << str.capacity() << endl; // 63
str += "string2 string3";
cout << str.size() << endl; // 22
cout << str.capacity() << endl; // 63
```

shrink_to_fit() funksiyasi satrning o'lchamini minimal qiymatgacha kamaytiradi. Metodning prototipi:

```
#include <string> void shrink_to_fit();
```

Misol:

```
string str("String");
str.reserve(50);
cout << str.capacity() << endl; // 63
str.shrink_to_fit();
cout << str.capacity() << endl; // 15
```

resize() – satrda Newsize soniga teng belgilar sonini beradi. Agar ko'rsatilgan belgilar soni joriy sondan kam bo'lsa, ortiqcha

belgilar o'chiriladi. Belgilar sonini ko'paytirish lozim bo'lsa, ch parametrida yangi sohani to'ldiruvchi belgini ko'rsatish mumkin. Metodning prototiplari:

#include <string>void resize(size_type Newsize);

void resize(size_type Newsize, char Ch);

Misol ko'raylik.

```
string str("String");
str.resize(4); cout << str << endl; // Stri
str.resize(8, '*'); cout << str << endl; // Stri****
```

clear() – barcha belgilarni o'chirish. Metodning prototipi:

#include <string> void clear();

Misol:

```
string str("String");
str.clear(); cout << str.size() << endl; // 0
```

empty() – satrda hech qanday belgi bo'lmasa true, aks holda false qaytaradi. Metodning prototipi:

#include <string> bool empty() const;

Misol:

```
string str("String");
if (str.empty()) {
 cout << "Bo'sh satr" << endl;
 } else {
 cout << "Satr belgilarni oladi" << endl;
 }
```

max_size() – satrda olish mumkin bo'lgan belgilarning maksimal sonini qaytaradi. Metodning prototipi:

#include <string> size_type max_size() const;

Misol:

```
string str("String");
cout << str.max_size() << endl; // 4294967294
```

Satr mazmunini olish va o'chirish. string sinfi obyektining ihtiyoriy belgisiga massivning elementiga kabi murojaat qilish mumkin. Buning uchun indeksini to'rtburchakli uchida

ko'rsatish yetarli. Nomerlash noldan boshlanadi. Bunda belgini ham olish, ham o'zgartirish mumkin. Agar indeks diapozondan chiqib ketsa, qaytariladigan qiymat aniqlanmaydi. Belgiga indeksi bo'yicha kirishga misol.

```
string str("Satr");
cout << str[0] << endl; // S
str[3] = 'D';
cout << str[3] << endl; // D
```

Satrdagi belgilar sonini olish uchun size() metodidan foydalaniladi. Misol sifatida satrdagi barcha belgilarni alohida satrda chiqarish dasturini keltirish mumkin:

```
string str("string");
for(int i = 0, c = str.size(); i < c; ++i) {
 cout << str[i] << endl;
}
```

string sinfi obyektlari uchun konkatenatsiya amali (ulash, satrlarni birlashtirish) aniqlangan. "+" operatori string sinfining ikkita obyektini birlashtirish imkonini beradi:

```
string str1("A"), str2("B"), str3; // string sinfidagi ikkita obyekt
str3 = str1 + str2; // AB
string str1("A"), str2, str3; // string sinfidagi obyekt bilan belgi
char ch = 'B';
str2 = str1 + ch; // AB
str3 = ch + str1; // BA
string str1("A"), str2, str3; // string sinfidagi obyekt bilan C-satr
str2 = str1 + "B"; // AB
str3 = "B" + str1; // BA
```

Ikkita C-satrni birlashtirish uchun "+" operatorini qo'llab bo'lmaydi. Ikkita C-satrni birlashtirish uchun ular operatorlarsiz ko'rsatiladi. Misol:

```
string str1("A"), str2, str3;
str2 = "C" + "B"; // Xato
str2 = "C" + str1 + "B"; // Normal holat
str3 = "C" "B"; // Normal holat – C-satrni birlashtirish
```

"+" operatoridan tashqari "+=" operatoridan ham foydalanish mumkin. Bu operator o'zlashtirish bilan konkatenatsiya amalini bajaradi.

```
string str1("A"), str2("B");
str1 += "C"; //AC
str2 += str1; // BAC
str2 += str1 + "D"; // BACACD
```

Satr mazmunini olish va o'zgartirish uchun quyidagi metodlardan foydalaniladi:

at() *metodi* off indeksi bo'yicha joylashgan belgiga murojaatni qaytaradi. Metod belgini ham olish, ham o'zgartirish imkonini beradi. Agar indeks diapozon chegarasidan chiqib ketsa, istesnoni generatsiya qiladi. Metodning prototiplari:

```
#include <string>
reference at(size_type Off);
const_reference at(size_type Off) const;
```

Misol:

```
string str1("String1");
const string str2("String2");
basic_string<char>::reference rstr1 = str1.at(0);
basic_string<char>::const_reference rstr2 = str2.at(6);
rstr1 = 's';
str1.at(6) = '3';
cout << str1 << endl; // string3
cout << str1.at(1) << endl; // t
cout << rstr2 << endl; // 2 (at() metodi)
cout << str2[6] << endl; // 2 ([] operatori)
```

front() *metodi* satrdagi birinchi belgiga murojaatni qaytaradi. Metod belgini olish, hamda o'zgartirish imkonini beradi. Metodning prototiplari:

#include <string>
reference front();
const_reference front() const;
Misol:
string str ("String");
str.front() = 's'; cout << str.front() << endl; // *s*

back() *metodi* satrdagi oxirgi belgiga murojaatni qaytaradi. Metod belgini olish, hamda o'zgartirish imkonini beradi. Metodning prototiplari:

#include <string> reference back() ;
const_reference back() const;
Misol:
string str("String");
str.back() = ' G'; cout << str.back() << endl; // *G*

substr() *metodi* off indeksidan boshlab Count belgilardan tashkil topgan satr qismini qaytaradi. Agar boshlang'ich indeks berilmasa, u nolga teng deb olinadi. Agar uzunlik berilmasa, off indeksidan boshlab satr oxirigacha bo'lgan parcha qaytariladi. Metodning prototipi:

#include <string>
string substr(size_type off=0, size_type Count=npos) const;
Misol:
string str1("12345"), str2;
str2 = str1.substr(); cout << str2 << endl; // *12345*
str2 = str1.substr(2); cout << str2 << endl; // *345*
str2 = str1.substr(2,2); cout << str2 << endl; // *34*

push_back() *metodi* Ch belgini satr oxiriga qo'shadi. Metodning umumiy ko'rinishi quyidagicha:

#include <string>
void push_back(char Ch);
Misol:
string str("String"); str.push_back('1') ;
cout << str << endl; // *String1*

append() *metodi* belgilar va satrlarni dastlabki satr oxiriga qo'shadi. Metoddan konkatenatsiya uchun "+" va "+=" operatorlari o'rniga foydalanish mumkin. Metodning prototiplari:

```
#include <string>
string &append(size type N, char Ch);
string &append(const char *Ptr);
string &append(const char *Ptr, size type N);
string &append(const string &Right);
string &append(const string &Right,size type Roff,size type N);
template<class It>
string &append(_It First, _It Last);
string &append(const_pointer First, const_pointer Last);
string &append(const_iterator First, const_iterator Last);
```

Birinchi prototip N ta Ch belgini satr oxiriga qo'shadi. Ikkinchi prototip C-satrni butunligicha, uchinchisi esa C-satrdan N ta birinchi belgini qo'shish imkonini beradi. To'rtinchi prototip string sinfidagi obyektni butunligicha, beshinchi prototip indeksidan boshlab N ta belgi qo'shish imonini beradi. Qolgan prototiplar iteratorlar yordamida boshlang'ich (First) va oxirgi (Last) pozitsiyalarni ko'rsatish imkonini beradi. Misollar:

```
string str1("A"), str2("BC");
char str3[] = "DEF";
str1.append(2, '*'); // Prototip 1
cout << str1 << endl; // A**
str1.append(str3); // Prototip 2
cout << str1 << endl; // A**DEF
str1.append(str3, 2); // Prototip 3
cout << str1 << endl; // A**DEFDE
str1 = "A";
str1.append(str2); // Prototip 4
cout << str1 << endl; // ABC
str1.append(str2, 0, 1); // Prototip 5
```

```
cout << str1 << endl; // ABCB
str1.append(str2.begin(), str2.end()); // Prototip 6
cout << str1 << endl; // ABCBBC
```

insert() *metodi* belgi va satrlarni indeks yoki iterator (ko'rsatkich) orqali ko'rsatilgan pozitsiyadan qo'yadi. Qolgan elementlar satrning oxiriga suriladi. Metodning prototiplari:

```
#include <string>
string.insert(size_type Off, size_type N, char Ch); // 1
iterator insert(const_iterator Where, char Ch); // 2
void insert(const_iterator Where, size_type N, char Ch); // 3
string.insert(size_type Off, const char *Ptr); // 4
string.insert(size_type Off, const char *Ptr, size_type N);//
5
string.insert(size_type Off,conststst string &Right); // 6
string.insert(size_type Off,const string &Right, size_type
Roff, size type N); // 7
template<class It>
void insert(const iterator Where, It First, It Last); // 8
void insert(const iterator Where, const pointer First, const
pointer Last); // 9
void insert(const iterator Where, const iterator First, const
iterator Last); // 10
iterator insert(const iterator Where); // 11
```

1-prototip Off indeksli pozitsiyadan boshlab N ta Ch belgsini qo'yadi.

2-prototip where o'zgarmas iteratori ko'rsatgan pozitsiyaga Ch belgsini qo'yadi. 3-prototip where o'zgarmas iteratori ko'rsatgan pozitsiyaga N ta Ch belgsini qo'yadi. 4-prototip Off indeksli pozitsiyaga Ptr C-satrini yaxlitligicha qo'yadi, 5-prototip esa Ptr C-satridan birinchi N ta belgini qo'yish imkonini beradi. 6-prototip Right string sinfi obyektini Off indeksli pozitsiyaga yaxlitligicha,7-prototip esa Roff indeksidan boshlab faqat N ta belgini qoyish imkonini beradi. 8-, 9- va 10-prototip-

lar iteratorlar (ko'rsatkichlar) yordamida boshlang'ich (First) va oxirgi (Last) pozitsiyalarni ko'rsatish imkonini beradi. Belgilar where o'zgarmas iteratori ko'rsatgan pozitsiyaga qo'yiladi. 11-prototip where o'zgarmas iteratori ko'rsatgan pozitsiyaga nol belgini qo'yadi. Misollar ko'ramiz.

```
string str1("12345"), str2("###");
char str3[]="+++";
str1.insert(2, 5, '*'); cout << str1 << endl; // Pr.1:
12*****345
str1 = "12345";
str1.insert(str1.begin() + 2, '*'); // Pr.2
cout << str1 << endl; // 12*345
str1 = "12345";
str1.insert(str1.begin() + 2, 5, '*'); // Pr.3
cout << str1 << endl; // 12*****345
str1 = "12345";
str1.insert(2, str3); // Pr.4
cout << str1 << endl; // 12+++345
str1 = "12345";
str1.insert(2, str3, 1); // Pr.5
cout << str1 << endl; // 12+345
str1 = "12345";
str1.insert(2, str2); cout << str1 << endl; // Pr.6, 12###345
str1 = "12345";
str1.insert(2, str2, 0, 1); cout << str1 << endl; // Pr.7,
12#345
str1 = "12345";
str1.insert(str1.begin() + 2, str2.begin(), str2.end()); // Pr.8
cout << str1 << endl; // 12###345
```

pop_back() *metodi* satrda oxirgi belgini o'chiradi. Metodning prototipi:

```
#include <string>
void pop_back();
```

Misol:

```
string str("12345");
str.pop_back();
cout << str << endl; // 1234
str.pop_back();
cout << str << endl; // 123
```

erase() *metodi* bir belgi yoki parchani o'chiradi. Qolgan belgilar satr boshiga suriladi. Metodning prototipi:

```
#include <string>
iterator erase(const_iterator Where);
iterator erase(const_iterator First, const_iterator Last);
string erase(size_type 0ff=0, size_type Count=npos);
```

Birinchi prototip where iteratori (ko'rsatkich) ko'rsatayotgan belgini o'chiradi. Ikkinchi prototip First va Last iteratorlari orasidagi belgilar qismini o'chiradi. Uchunchi prototip off indeksidan boshlab Count uzunlikdagi parchani o'chiqadi. Agar boshlang'ich indeks ko'rsatilmasa, u nolga teng deb qabul qilinadi. Uzunlik ko'rsatilmasa parcha off indeksidan boshlab, satr oxirigacha o'chiriladi. Misollar:

```
string str("12345");
str.erase(str.begin() + 2); cout << str << endl;
str="12345";
str.erase(str.begin(), str.begin() + 2); // Prototip 2
cout << str << endl;
str="12345";
str.erase(3); cout << str << endl;
str="12345";
str.erase(0, 2); cout << str << endl;
```

swap() – ikkita satr mazmuni o'rnini almashtiradi. Metodning prototipi:

```
#include <string>
void swap(string &Right) ;
```

Misol:

```
string str1("12345"}, str2("67890");
str1.swap(str2);
cout<<str1<<endl; // 67890
cout<<str2<<endl; // 12345
```

replace() – satr qismini alohida berlgi yoki satr osti bilan almashtiradi. Agar qo'yilayotgan satr osti parchadan kichik bo'lsa, qolgan belgilar satr boshiga suriladi, katta bo'lsa, qo'yilayotgan satr osti sig'ishi uchun suriladi. Funksiyaning prototiplari:

```
#include <string>
string.replace(size_type Off, size_type N0,
 size_type K, char Ch);
string.replace(const_iterator First, const_iterator Last,
 size_type K, char Ch) ;
string.replace(size_type Off, size_type N0, const char *Ptr);
string.replace(size_type Off, size_type N0,
 const char *Ptr, size_type K);
string.replace(const_iterat or First, const_iterator Last,
 const char *Ptr);
string.replace(const_iterat or First, const_iterator Last,
 const char *Ptr, size_type K);
string.replace(size_type Off, size_type N0,
 const string &Right);
string.replace(size_type Off, size_type N0,
 const string &Right, size_type Roff, size_type K);
string.replace(const_iterat or First, const_iterator Last,
 const string &Right);
```

1-prototip Off indeksidan boshlab N0 uzunlikdagi qism o'rniga K ta Ch belgini qo'yadi. 2-prototip First va Last iteratorlari ko'rsatayotgan elementlar bilan chegaralangan qism o'rniga K ta Ch belgini qo'yadi. 3-prototip Off indeksidan boshlanuvchi N0 uzunlikdagi qism o'rniga Ptr C-satrini yaxlitligicha, 4-prototip esa Ptr C-satridan faqat birinchi K tasini qo'yish imkonini beradi. 5-prototip First va Last iteratorlari ko'rsatayotgan

elementlar bilan chegaralangan qismning o'rniga Ptr C-satrini yaxlitligicha qo'yadi. 6-prototip esa faqat birinchi K ta belgini. 7-prototip Off indeksidan boslanuvchi N0 uzunlikdagi qism o'rniga Right string sinfi obyektini, 8-prototip esa Roff indeksidan boshlab faqat K ta belgini qo'yish imkonini beradi. 9-prototip First va Last iteratorlari ko'rsatayotgan elementlar bilan chegaralangan qism o'rniga Right string sinfi obyektini to'laligicha qo'yish imkonini beradi. Misollar ko'ramiz:

```
string str1("12345"), str2("67890");
char str3[]="abcd";
str1.replace(0, 3, 4, '*'); // Pr.1
cout << str1 << endl; // * * * * 4 5
str1.replace(str1.begin(), str1.begin() + 3, 3, '+'); // Pr.2
cout << str1 << endl; // ++++45
str1.replace(0, 4, str3); // Pr.3
cout << str1 << endl; // abcd45
str1.replace(0, 4, str3, 3); // Pr.4
cout << str1 << endl; // abc45
str1.replace(str1.begin(), str1.begin() + 3, "0123"); // Pr.5
cout << str1 << endl; // 012345
str1.replace(str1.begin(), str1.begin() + 1, str3, 2); // Pr.6
cout << str1 << endl; // ab12345
str1.replace(0, 5, str2); // Pr.7
cout << str1 << endl; // 6789045
str1.replace(0, 5, str2, 1, 2); // Pr.8
cout << str1 << endl; // 7845
str1.replace(str1.begin(), str1.begin(), str2); // Pr.9
cout << str1 << endl; // 678907845
```

Satrda qidirish uchun tayinlangan metodlarni keltirib o'tamiz:

find() metodibelgi yoki qismni satrning boshidan boshlab qidirishni amalga oshiradi. Agar qism toplilsa, birinchi mos kelishning indeksini, aks holda string::npos o'zgarmasining qi-

ymatini qaytaradi. Qidiruv belgilar registrida bog'liq. Metodning prototiplari:

```
#include <string>
size_type find(char Ch, size_type Off=0) const;
size_type find(const char *Ptr, size_type Off=0) const;
size_type find(const char *Ptr, size_type Off, size_type N)
const;
size_type find(const string &Right, size_type Off=0) const;
```

Birinchi prototip satrda off indeksdan boshlab ch belgini qidiradi. Ikkinchi prototip C-satrni off indeksdan boshlab to'laligicha, uchinchisi esa C-satrning faqat birinchi N ta belgisini qidirish imkonini beradi.To'rtinchi prototip string sinfidagi obyektni off indeksidan boshlab qidiradi. Agar indeksi berilmagan bo'lsa, uning qiymati "0" ga teng deb qabul qilinadi.

rfind() metodi belgi yoki matn qismini satr oxiridan boshlab qidiradi. Agar qism topilmasa, oxiridan boshlab birinchi moslikning indeksini, aks holda std::string::npos o'zgarmasining qiymatini qaytaradi. Qidiruv belgilar registriga bog'liq. Metodning prototippplari:

```
#include <string>
size_type rfind(char Ch, size_type Off=npos) const;
size_type rfind(const char *Ptr, size_type Off=npos) const;
size_type rfind(const char *Ptr, size_type Off, size_type N)
const;
size_type rfind(const string &Right, size_type Off=npos)
const;
```

Birinchi prototip satrda off indeksidan boshlab ch belgisini qidiradi. Ikkinchi prototip C-satrni off indeksidan butunligicha qidirish imkonini beradi, uchinchi prototip – C-satrning faqat birincihi N ta belgisini. To'rtinchi prototip off indeksidan boshlab string sinfi obyektini qidiradi. Qidiruv off indeksidan satr boshiga qarab amalga oshiriladi. Agar off indeksi ko'rsatilmasa, indeks qiymati oxirgi element qiymatiga teng deb qabul qilinadi.

```
string str("123454321"), str2("2");
cout << str.rfind('2') << endl; // 7
cout << (int)str.rfind('6') << endl; // -1
cout << str.rfind('23) << endl; // 1
cout << str.rfind("2") << endl; // 7
cout << (int)str.rfind("6") << endl; // -1
cout << str.rfind("2", 3) << endl; // 1
cout << str.rfind("200", 3, 1) << endl; // 1
cout << str.rfind(str2) << endl; // 7
cout << str.rfind(str2, 3) << endl; // 1
```

find_first_of() metodi qandaydir belgini yoki ko'rsatilgan satrga kiruvchi qandaydir belgini qidirish uchun tayinlangan. Qidirish satrning boshidan boshlanadi. Metod belgi topilsa, birinchi mos kelishning indeksini, aks holda string::npos o'zgarmasining qiymatini qaytaradi. Qidirish belgilar registriga bog'liq. Metodning prototiplari:

```
#include <string>
size_type find_first_of(char Ch, size_type Off=0) const;
size_type find_first_of(const char *Ptr, size_type Off=O) const;
size_type find_first_of(const char *Ptr, size_type Off,
size_type Count) const;
size_type find_first_of(const string &Right, size_type
0ff=0) const;
```

Birinchi prototip satrda off indeksidan boshlab ch belgisini qidiradi. Ikkinchi prototip off indeksidan C-satrdagi qandaydir belgi bilan moslikni qisiradi. Uchinchi prototip C-satrdagi faqat birincihi Cout belgini. To'rtinchi prototip off indeksidan boshlab string sinfi obyekti yoki biror belgi bilan moslikni qidiradi. Agar off indeksi berilmagan bo'lsa, uning qiymati "0" ga teng deb qabul qilinadi.

Misollar:

```
string str ("123454321") , str2 ("862 ") ;
cout << str.find_first_of('2') << endl;
```

```
cout << (int)str.find_first_of('6') << endl;
cout << str.find_first_of('2', 3) << endl;
cout << str.find_first_of("862") << endl;
cout << str.find_first_of("862", 3) << endl;
cout << str.find_first_of("250", 3, 2) << endl;
cout << str.find_first_of(str2) << endl; // 1
cout << str.find_first_of(str2, 3) << endl; // 7
```

find_first_not_of() metodi ko'rsatilgan satrning hech bir belgisi bilan mos kelmaydigan birinchi belgining indeksini qaytaradi. Agar hech narsa topilmasa, std::string::npos o'zgarmasining qiymatini qaytaradi. Qidiruv registrga bog'liq. Metodning prototiplari:

```
#include <string>
size_type find_first_not_of(char Ch, size_type 0ff=0) const;
size type find first not of(const char *Ptr, size_type Off=0)
const;
size_type find_first_not_of(const char *Ptr, size_type Off,
size_type N)
const; size_type find_first_not_of(const string &Right,
size_type 0ff=0) const;
```

Birinchi prototip satrda ch belgisi bilan mos kelmaydigan belgini qidiradi. Ikkinchi prototip Ptr C-satrga kiruvchi hech bir belgi bilan mos kelmaydigan birinchi belgining indeksini qaytaradi, uchinchi prototip esa Ptr C-satrning birinchi N ta belgisini hisobga oladi. To'rtinchi prototip string sinfi obyektiga kiruvchi hech bir belgi bilan mos kelmaydigan birinchi belgining indeksini qaytaradi. Qidiruv off indeksidan boshlanadi. Indeks ko'rsatilmagan bo'lsa, uning qiymati '0'ga teng deb olinadi. Misollar:

```
string str("123454321"), str2 ("8621");
cout << str.find_first_not_of('2') << endl; // 0
cout << str.find_first_not_of('2', 3) << endl; // 3
cout << str.find_first_not_of("8621") << endl; // 2
```

```
cout << str.find_first_not_of("8621", 3) << endl; // 3
cout << str.find_first_not_of("250 ", 3, 2) << endl; // 3
cout << str.find_first_not_of(str2) << endl; // 2
cout << str.find_first_not_of(str2, 3) << endl; // 3
```

find_last_of() metodi find_first_of() metodiga o'xshaydi, ammo qidiruv satr boshidan emas, balki oxiridan boshlanadi. Metodning prototipi:

```
#include <string>
size_type find_last_of(char Ch, size_type Off=npos) const;
size_type find_last_of(const char *Ptr, size_type Off=npos) const;
size_type find_last_of(const char *Ptr, size_type Off, size_type N) const;
size_type find_last_of(const string &Right,size_type Off=npos) const;
```

find_last_not_of() metodi find_first_not_of() metodi bilan bir xilda, ammo qidiruv satr boshidan emas, balki oxiridan boshlanadi.

Metodning prototipi:

```
#include <string>
size_type find_last_not_of(char Ch, size_type Off=npos) const;
size_type find_last_not_of(const char *Ptr, size_type Off=npos) const;
size_type find_last_not_of(const char *Ptr, size_type Off,size_type N) const;
size_type find_last_not_of(const string &Right, size_type Off=npos) const;
```

Misol tariqasida satr faqat sonlardan tashkil topganmi yoki yo'qligini (harflar mavjudlibini) tekshiramiz:

```
string str("5487");
int index = 0;
index = str.find_last_not_of("0123456789");
```

```
}
else cout << "Yo'q" << endl;
```

Satrlarni taqqoslash. Ikkita string sinfidagi obyektni yoki string sinfidagi obyekt va C-satrni taqqoslash uchun "==" (teng), "!=" (teng emas), "<" (kichik), ">" (katta), "<=" (kichik yoki teng) va ">=" (katta yoki teng) operatorlaridan foydalaniladi. Satrlarni taqqoslashga doir misol keltirib o'tamiz:

```
string str1("abc"), str2("abed");
if(str1 == str2) {
cout << "str1 == str2" << endl;
} else if(str1 < str2) {
cout << "str1 < str2" << endl;
} else if(str1 > str2) {
cout << "str1 > str2" << endl;
} // Natija: str1 < str2
```

Bu operatorlar yordamida C-satrlarni taqqoslab bo'lmaydi, chunki satrdagi belgilar emas, balki ularning adresi (manzili) taqqoslanadi. C-satrlarni taqqoslash uchun maxsus funksiyalardan foydalanish zarur, masalan, strcmp(). Operatorlar korrekt ishlashi uchun operatordan chap yoki o'ng tomonda string sinfi obyekti joylashishi zarur. Misol:

```
char a[] = "abc", b[] = "abc";
string c("abc");
cout << (a == b) << endl; // Manzillar taqqoslanadi
cout << strcmp(a, b) << endl; // teng)
cout << (a == c) << endl; // teng
cout << (c == b) << endl; // teng
```

string sinfidagi obyektni boshqa string sinfidagi obyekt yoki C-satr bilan taqqoslash uchun compare() metodidan foydalanish mumkin. Metodning prototiplariga qisqachga yo'xtalib o'tamiz.

Prototip 1: string sinf obyektini C-satr bilan taqqoslaydi;

```
#include <string>
```

int compare(const char *Ptr) const;

Misol:

string str1("abc"), str2("abd");

cout << str1,.compare("abd") << endl;

Prototip 2: string sinfi obyektidagi off indeksidan boshlab N0 uzunlikdagi parchani C-satr bilan taqqoslaydi;

#include <string>

int compare(size_type Off, size_type N0, const char *Ptr) const;

Misol:

string str1("abc"), str2("abd");

cout << str1,.compare(0, 2, "ab") << endl;

Prototip 3: C-satrda satr boshidan boshlab qo'shimcha ravishda parcha uzunligini ko'rsatish imkonini beradi;

#include <string>

int compare(size_type Off, size_type N0, const char *Ptr, size_type Count) const;

Misol:

string str1("abc"), str2("abd");

cout << strl,.compare(0, 2, "abc", 2) << endl;

Prototip 4: string sinfidagi ikkita obyektni taqqoslaydi;

#include <string>

int compare(const string &Right) const;

Misol:

string str1("abc"), str2("abd");

cout << str1,.compare(str2) << endl;

Prototip 5: string sinfi obyektidagi off indeksidan boshlab N0 uzunlikdagi parchani boshqa string sinfi obyekti bilan taqqoslaydi;

#include <string>

int compare(size_type Off, size_type N0, const string &Right) const;

Misol:

```
string str1("abc"), str2("abd");
cout << str1,.compare(0, 2, str2) << endl;
```

Prototip 6: Right satrida qo'shimcha ravishda uzunlik (Count) va boshlang'ich indeksni (Roff) ko'rsatish imkonini beradi. Taqqoslash registrni hisobga olgan holda bajariladi. qiymat sifatida quyidagilardan birini qaytaradi: manfiy son – string sinfi obyekti parametr sifatida uzatilgan satrdan kichik bo'lsa, 0 – agar satrlar teng bo'lsa va musbat son – string sinfi obyekti parametr sifatida uzatilgan satrdan katta bo'lsa.

```
#include <string>
int compare(size_type Off, size_type N0, const string
&Right, size_type Roff, size_type Count) const;
```

Misol:

```
string str1("abc"), str2("abd");
cout << str1,.compare(0, 2, str2, 2) << endl;
```

9.3. Satrni songa va sonni satrga almashtirish

Satrni songa almashtirish uchun quyidagi funksiyalardan foydalaniladi:

atoi() va atol() funksiyalari C-satrni mos ravishda int va long tipidagi sonlarga almashtiradi. Funksiyalarning prototiplari:

```
#include <cstdlib>
int atoi(const char *Str);
long atol(const char *Str);
```

Belgilarni o'qish raqamlarga mos kelganda amalga oshiriladi. Ya'ni, satrda raqamdan tashqari boshqa belgilar ham bo'lishi mumkin. Satr boshidagi bo'shliq (пробель) e'tiborga olinmaydi. Almashtirishga misollar:

```
cout << atoi("25") << endl; // 25
cout << atoi("2.5") << endl; // 2
cout << atoi("5str") << endl; // 5
cout << atoi("5s10") << endl; // 5
```

```
cout << atoi("\t\n 25") << endl; // 25
cout << atoi("-25") << endl; // -25
cout << atoi("str") << endl; // 0
```

atol() funksiyalari ham shunday natija beradi.

strtol() va strtoul() – C-satrni mos ravishda long va unsigned long tipidagi songa almashtiradi. Funksiyalarning prototiplari:

```
#include <cstdlib>
long strtol(const char *Str, char **EndPtr, int Radix);
unsigned long strtoul(const char *Str, char **EndPtr, int Radix);
```

Satr boshidagi bo'shliq belgilari e'tiborga olinmaydi. Belgilarni o'qish son yozuvi bo'lmagan belgida tugallanadi. Bu belgiga ko'rsatkich, agar u nolga teng bo'lmasa, EndPtr parametr orqali ochiq. Radix parametrida sanoq sistemasini ko'rsatish mumkin (2 dan 36 gacha son). Agar Radix parametrida '0' soni berilgan bo'lsa, u holda sanoq sistemasi avtomatik ravishda aniqlanadi. Almashtirishga misollar:

```
char *P = 0;
cout << strtol("25", 0, 0) << endl; // 25
cout << strtol("025", 0, 0) << endl; // 21
cout << strtol("0x25", 0, 0) << endl; // 37
cout << strtol("111", 0, 2) << endl; // 7
cout << strtol("025", 0, 8) << endl; // 21
cout << strtol("0x25", 0, 16) << endl; // 37
cout << strtol("5s10", &P, 0) << endl; // 5
cout << P<< endl; // s10
system("Pause");
```

Agar satrda son bo'lmasa, nol soni qaytariladi. Agar son tip uchun qiymatlar diapozonidan chiqib ketsa, u holda qiymat tip uchun minimal (LONG_MIN yoki 0) yoki maksimal (LONG_MAX yoki ULONG_MAX) qiymatga mos bo'ladi. errno o'zgaruvchisiga ERANGE qiymati o'zlashtiriladi. Xatolik bayroqchasini tashlash uchun errno o'zgaruvchisiga '0' qiymat o'zlashtiriladi. Misol:

```
cout << strtol("str", 0, 0) << endl; // 0
cout << strtol("99999999999999999", 0, 0) << endl;
// 2147483647 (LONG_MAX ga mos keladi)
if(errno == ERANGE) {
 cout << "Xato" << endl; // Error
 }
errno = 0; // xatolik bayroqchasini tashlash
cout << strtol("25", 0, 0) << endl; // 25
if(errno != ERANGE) {
 cout << "OK" << endl; // OK
 }
system("Pause");
```

```
0
2147483647
Xato
25
OK
Для продолжения нажмите любую клавишу . . .
```

atof() – C-satrni double tipidagi songa almashtiradi. Funksiyaning prototipi:

```
#include <cstdlib>
double atof(const char *String);
cout << atof("25") << endl; // 25
cout << atof("2.5") << endl; // 2.5
cout << atof("5.1str") << endl; // 5.1
cout << atof("5s10") << endl; // 5
cout << atof("str") << endl; // 0
```

Satr boshidagi bo'shliq belgilari e'tiborga olinmaydi. Belgilarni o'qish haqiqiy son yozuvi bo'lmagan belgida tugallanadi.

Bu belgiga ko'rsatkich, agar u nolga teng bo'lsa, EndPtr parametri orqali kirishli (ochiq). Almashtirishga misol: char *p=0;

strtod() – C-satrni double tipiga ega bo'lgan songa almashtiradi. Funksiyaning prototipi:

```
#include <cstdlib>
double strtodfconst char *Str, char **EndPtr);
cout << strtod("\t\n 25",0) << endl; // 25
cout << strtod("2.5",0) << endl; // 2.5
cout << strtod("5.1str",0) << endl; // 5.1
cout << strtod("14.5e5s10",&p) << endl; // 1.45e+006
cout << p << endl; // s10
```

Agar satrda son bo'lmasa, u holda '0' soni qaytariladi. Son tip diapozonidan chiqib ketsa, u holda – UGE_VAL yoki HUGE_VAL, qiymati qaytariladi, errno o'zgaruvchisiga esa, ERANGE qiymat o'zlashtiriladi. Xatolik bayroqchasini tashlash uchun errno o'zgaruvchisiga '0' qiymat o'zlashtirish lozim.

Ko'rib o'tilgan barcha funksiyalar butun C-satrni songa aylantirish imkonini beradi. Alohida belgini (sonni ifodalovchi) mos ravishda 0 dan 9 gacha butun songa almashtirish zarur bosa, quyidagi koddan foydalanish mumkin: char ch = '8';

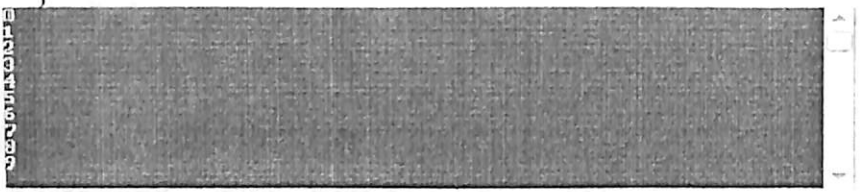
```
int n = ch – '0'; // int n=56–48 ga ekvivalent
cout << n << endl; // 8
```

char tipidagi o'zgaruvchida belgi emas, uning kodi saqlanadi. ASCII kodlashida '0' dan '9' gacha belgilar mos ravishda 48 dan 57 gacha kodlarga ega. Bundan kelib chiqadiki, '0' dan '9' gacha butun sonni olish uchun joriy belgidan '0' belgi kodini ayirish yetarli. Yana bir misol sifatida C-satrda uchrovchi barcha sonlarni butun qiymatli massivda saqlashni ko'ramiz.

C-satr belgilarini alohida sonlarga almashtirish:

```
#include <iostream>
```

```
int main() {
 char str[] = "0123456789";
 const int SIZE = 10;
 int mas[SIZE] = {0}, index = 0;
 for(char *p = str; *p; ++p) { // Satrni birma-bir ko'rish
 if(*p >= '0' && *p <= '9') {
 mas[index] = *p – '0';
 ++index;
 if(index >= SIZE) break;
 }
 }
 for(int i = 0; i < SIZE; ++i) { // Qiymatlarni chiqarish
 cout << mas[i] << endl;
 }
 cin.get();
 return 0;
}
```



Sonni satrga almashtirish. sprintf() funksiyasi elementar tiplar qiymatlarini C-satrga almashtirish imkonini beradi. Funksiyaning prototipi:

```
#include <cstdio>
int sprintf(char *DstBuf, const char *Format)
```

Format parametrida maxsus formatdagi satr ko'rsatiladi. Bu satrning ichida oddiy belgilar va "%" belgisidan boshlanuvchi format spesifikatorlarini ko'rsatish mumkin. Format spesifikatorlari printf() funksiyasida foydalaniluvchi spesifikatorlar blan mos keladii.

Format spesifikatorlari o'rniga parametr sifatida ko'rsatilgan qiymatlar qo'yilaid. Spesifikatorlar soni uzatilgan parametrlar

berilgan (DstBuf) buferga yoziladi. Funksiya qiymat sifatida belgili massivga yozilgan belgilar sonini qaytaradi.

Butun sonni C-satrga almashtirishga misol:

```
char buf[50];
int x = 100, count = 0;
count = sprintf(buf, "%d", x);
cout << buf << endl; // 100
cout << count << endl; // 3
```

sprintf() funksiyasi bufer o'lchamini tekshirmaydi, shu sababli bufer to'lib ketishi mumkin. VC++da sprintf() funksiyasi o'rniga sprintf_s() funksiyasidan foydalangan ma'qul, funksiyaning prototipi:

```
#include <stdio.h>
int sprintf_s(char *DstBuf, size_t SizeInBytes, const char
*Format)
```

DstBuf va Format parametrlari sprintf() funksiyasi parametrlariga mos. SizeInBytes parametrida bufer o'lchami ko'rsatiladi. Funksiya qiymat sifatida belgilar massiviga yozilgan belgilar sonini qaytaradi. Haqiqiy sonni C-satrga almashtirishga misol:

```
char buf[50]; int count = 0;
double pi = 3.14159265359;
count = sprintf_s(buf, 50, "'110.5f'", pi);
cout << buf << endl; // '3.14159'
cout << count << endl; // 12
```

9.4. Tuzilmalar

Tuzilma (Struktura) – bu bir nom bilan birlashtirilgan o'zgaruvchilar (a'zolar, elementlar yoki maydonlar deb nomlanuvchi) mosligidir. Strukturani e'lon qilish umumiy holda quyidagicha ko'rinishga ega:

```
struct [<Struktura nomi>] {
<Berilganlar tipi><Maydon nomi_1>; ...;
```

<Berilganlar tipi><Maydon nomi_N>;

} [<O'zgaruvchilar e'loni(vergul bilan ajratilgan)>];

Agar yopuvchi figurali qavsdan keyin o'zgaruvchi e'lon qilingan bo'lsa, strukturasning nomini bermaslik ham mumkin. Sturktura e'loni oxirida nuqtali vergul qo'yish majburiy.

Tuzilmani e'lon qilish berilganlarning yangi tipini ifodalaydi xalos, shu sababli unga xotira ajratilmaydi. O'zgaruvchini e'lon qilihh uchun uning nomi yopiluvchi qavsdan keyin qo'yiladi yoki berilganlarning tipi sifatida struktura nomi ko'rsatilib, alohida e'lon qilinadi.

[struct]<Struktura nomi><O'zgaruvchilar nomi>;

O'zgaruvchini e'lon qilishda struct kalit so'zini ko'rsatmaslik ham mumkin. O'zgaruvchi e'lon qilinganidan keyin kompilyator zarur xotira hajmini ajratadi. Dastur ichida struktura o'lchamini olish uchun sizeof operatoridan foydalanish lozim:

<O'lcham> = sizeof<O'zgaruvchi>;

<O'lcham> = sizeof(<Struktura nomi>);

Struktura va o'zgaruvchini bir vaqtda e'lon qilishga misol keltiramiz.

```
struct Point{
 int x; int y;
} point1;
```

O'zgaruvchini alohida e'lon qilish quyidagicha ko'rinishda bo'ladi:

Point point2;

Maydonga qiymat berish yoki qiymatni olish uchun nuqta notatsiyasidan (*shartli nuqta*) foydalanish mumkin:

<O'zgaruvchi>.<Maydon nomi> = <Qiymat>;

<Qiymat> = <O'zgaruvchi>.<Maydon nomi>;

Misol: point1.x = 0; z = point2.y;

Bir strukturani "=" operatori yordamida boshqa strukturaga o'zlashtirish mumkin. Bunda strukturaning barcha maydoni nusxalanadi. Strukturaga strukturani o'zlashtirishga misol:

point2 = point1;

Strukturalarni ichma-ich ifodalash mumkin. Bunda ichki struktura maydoniga murojaat qilish uchun qo'shimcha ravishda ajdod struktura nomi ko'rsatiladi. Misol tariqasida to'g'ri to'rtburchakni ifodalovchi Point (nuqta) strukturasini nomsiz elon qilish dasturi listinggini keltirib o'tamiz.

```
#include <iostream>
struct Point { // Nomlangan strukturani e'lon qilish
 int x; int y;
};
struct { // Nomsiz strukturani e'lon qilish
 Point top_left;
 Point bottom_right;
 } rect;
int main() {
 rect.top_left.x = 0; rect.top_left.y = 0;
 rect.bottom_right.x = 100; rect.bottom_right.y = 100;
 cout << rect.top_left.x << "\t" << rect.top_left.y << endl;
        cout<<rect.bottom_right.x<<"\t"<<rect.bottom_right.
y<<endl;
 cin.get();
 return 0;
 }
```

```
0       0
100     100
```

Struktura manzilini ko'rsatkichda saqlash mumkin. Strukturaga ko'rsatkichning e'loni boshqa tiplarga ko'rsatkich kabi e'lon qilinadi. Struktura manzilini olish uchun "&" operatoridan foydalaniladi, struktura maydoniga kirish uchun nuqta o'rniga "->" operatori qo'llaniladi. Strukturaga ko'rsatkichdan foydalanishga misol quyidagi listingda keltirilgan.

```
#include <iostream>
struct Point { // Struktura va o'zgaruvchini e'lon qilish
```

```
int x; int y;
} point1;
Point *p = &point1; // Ko'rsatkichni e'lon qilish
int main() {
p->x = 10; p->y = 20;
cout << p->x << «\t» << p->y << endl;
cin.get();
return 0;
}
```

Bitli maydonlar. Berilganlarning mantiqiy tipi bool faqat true ('1') yoki false ('0') qiymat olishi mumkin. Bu qiymatlar bir bitga joylashadi, ammo bool tipi xotirada bir baytni band qiladi. Qolgan yeti bit qiymatlar olmaydi. C++ tili alohida bitlarga kirishni ta'minlovchi bitli maydonlarni ta'minlaydi va shu tariqa ko'rsatilgan sondagi bitlarni olgan bir o'zgaruvchida bir nechta qiymatlarni saqlash imkonini beradi. Shuga qaramasdan, bitli maydonning minimal o'lchami int ('32' bitli operatsion tizimda '4' bayt band qiladi) tipiga mos kelishini hisobga olish lozim. Bitli maydon e'loni:

```
struct[<Bitli maydon nomi>] {
<Berilganlar tipi>[<Maydon nomi1>]:<Uzunlik bitda>;
<Berilganlar tipi>[<Maydon nomiN>]:<Uzunlik bitda>;
} [<O'zgaruvchilar vergul bilan e'lon qilinadi>];
```

Bir strukturada bir vaqtda ham bitli, ham oddiy maydonlardan foydalanish mumkin. Bitli maydonning nomini ko'rsatish shart emas. Bundan tashqari maydonning uzunligi bir bitga teng bo'lsa, nomdan oldin unsigned kalit so'zini ko'rsatish lozim. Bitli maydon va o'zgaruvchini e'lon qilishga misol:

```
struct Status {
unsigned :3;
unsigned a:1;
unsigned b:1;
unsigned c:1;
```

} status;

Maydonga kirish struktura maydoniga kirish kabi amalga oshiriladi:

```
status.a = 1;
status.b = 0;
status.c = 1;
```

9.5. Birlashmalar

Birlashma – bu bir vaqtda turli tipdagi berilganlarni saqlash uchun foydalaniluvchi xotira sohasi. Birlashmaning o'lchami eng murakkab berilganlar tipiga mos keladi. Masalan, birlashmada int, float va double tipiga ega bo'lgan o'zgaruvchilar aniqlangan. U holda birmashmaning o'lchami double tipiga mos keladi.

Birlashmani e'lon qilish quyidagicha formatga ega:

union[<Birlashma nomi>] {
<Berilganlar tipi><A'zo nomi1>;
...
<Berilganlar tipi>< A'zo nomiN>;
} [<O'zgaruvchilar e'loni: vergul bilan ajratilgan>];

E'lon oxirida nuqtali vergulni ko'rsatish majburiy hisoblanadi. E'lon berilganlarning yangi tipni bayon qiladi, ammo o'zgaruvchini aniqlamaydi, shu sababli, u uchun xotira ajratilmaydi. O'zgaruvchining e'loni birlashma e'lonidan keyin yopiluvchi figurali qavsdan keyin yoki birlashma moni berilganlar tipi sifatida ko'rsatilib, alohida beriladi:

[union]<Birlashma nomi> <O'zgaruvchilar nomi>;

VC++da o'zgaruvchini e'lon qilishda union kalit so'zini ko'rsatmaslik ham mumkin. O'zgaruvchi e'lon qilinganidan keyin kompilyator xotiradan zarur o'lchamni ajratadi. Birlashma va o'zgaruvchuni e'lon qilishga misol:

```
union Uni {
 int x;
```

```
float y;
double z;
} union1;
```

Qiymatni olish yoki o'zlashtirish shartli nuqta (nuqta notatsiyasi) yordamida amalga oshiriladi:

<O'zgaruvchi>.<A'zo nomi> = <Qiymat>;

<Qiymat> = <O'zgaruvchi>.<A'zo nomi>;

Misol:

```
union1.x = 152;
cout << union1.x << endl; // 152
```

Birlashmalar funksiyalarni hamda konstruktor va destruktorlarni olishi mumkin. Funksiyalarnini e'lonini birlashma e'loni ichida, aniqlanishi esa e'londan tashqarida joylashtiriladi. Funksiya nomidan oldin birlashma nomi va "::" operatori ko'rsatiladi. Funksiya ichida birlashma a'zolariga birlashma nomini ko'rsatmasdan murojaat qilish mumkin.

Birlashmalardan foydalanishga misol.

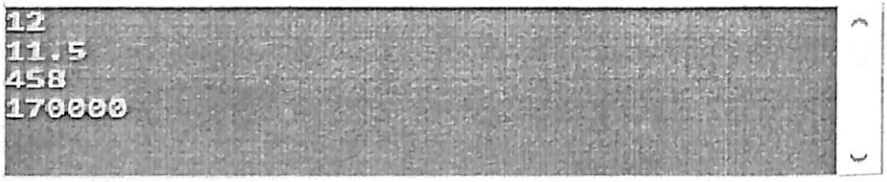
```
#include <iostream>
union Uni {
int x; double y;
void print_x();
void print_y();
};
void Uni::print_x() {
cout << x << endl;
}
void Uni::print_y() {
cout << y << endl;
}
int main() {
Uni union1;
union1.x = 152; union1.print_x(); // 152
union1.y = 1.5e-5; union1.print_y(); // 1.5e-005
```

```
cin.get();
return 0;
}
```

```
152
1.5e-005
```

Bir vaqtda birlashma nomini va o'zgaruvchilar e'lonini bermaslik ham mumkin. Bunday holda birlashma nomsziz yoki anonim deyiladi. Nomsiz birlashmalar a'zolariga kirish aynan a'zolar nomlari orqali amalga oshiriladi. Birlashma a'zolari nomlari ayni shu nomlar fazosida e'lon qilingan o'zgaruvchilar nomlari bilan to'qnashmasligi kerak. Global ko'rinish sohasida yoki nomlar fazosi ichida joylashgan nomsiz birlashmalar static kalit so'zi yordamia e'lon qilinishi lozim. Quyidagi listingda nomlanmagan (anonim) birlashmalardan foydalanish keltirilgan.

```
#include <iostream>
static union { // static so'zini ko'rsatish majburiy
 int x; double y;
} ;
int main() {
 union { // static so'zi ko'rsatilmaydi
 int a; double b;
 } ;
 x = 12; cout << x << endl; // 12
 y = 11.5; cout << y << endl; // 11.5
 a = 458; cout << a << endl; // 458
 b = 1.7e5; cout << b << endl; // 170000
 cin.get();
 return 0;
}
```



10. Sana va vaqt bilan ishlash

C++ tilida sana va vaqt bilan ishlash uchun tayimlahgan asosiy funksiyalar ctime (C tilida time.h fayli) sarlavha faylida berilgan. Faylda quyidagi tip ma'lumotlar ham aniqlangan.

size_t – strftime() funksiyasida ishlatiladi. O'lchami sozlashga bog'liq. Ko'rsatilmaganda 4 bayt. Tipning aniqlanishi quyidagicha:

typedef unsigned int size_t; typedef unsigned int64 size_t;

clock_t – clock() funksiyasi oraqli qaytariladi. Tipning aniqlanishi:

typedef long clock_t;

time_t funksiyasidan vaqtni butun qiymatli ko'rinishda tasvirlash uchun foydalaniladi. Tipning o'lchami sozlashlarga bog'liq. ko'rsatilmaganda 8 bayt band qiladi. Tipning aniqlanishi:

typedef time32_t time_t; typedef time64_t time_t;

time32_t va time64_t tiplarining aniqlanishi quyidagicha:

typedef long time32_t; typedef int64 time64_t;

Ba'zi funksiyalar qiymat sifatida "tm" strukturaga ko'rsatkich qaytaradi. "tm" strukturasining aniqlanishi quyidagicha ko'rinishga ega:

Struct tm {

int tm_sec; *// Sekundlar (0 dan 59 gacha, ayrim hollarda 61 gacha)*

int tm_min; *// Minutlar (0 dan 59 gacha son)*

int tm_hour; *// Soat (0 dan 23 gacha son)*

int tm_mday; *// Oy kuni (1 dan 31 gacha son)*

int tm_mon; *// Oy (0 (yanvar) dan 11 (dekabr) gacha)*

```
int tm_year; // Yil, 1900-yildan boshlab
int tm_wday; // Hafta kuni (0 dan 6 gacha son
//(0-bu yakshanba, 1-dushanba, ..., 6-shanba))
int tm_yday; // Yildagi kun (0 dan 365 gacha son)
int tm_isdst; // 0 dan katta bo'lsa, yozgi vaqt amalda.
// 0 bo'lsa, demak yozgi vaqt o'rnatilmagan,
// 0 dan kichik bo'lsa, ma'lumot yo'q
};
```

10.1. Joriy sana va vaqtni olish

Kompyuterda joriy sana va vaqtni olish uchun quyidagi funksiyalardan foydalaniladi:

time() funksiyasi kompyuter davrining boshlanishidan boshlab o'tgan sekundlar sonini qaytaradi (1970-yil 1-yanvardan boshlab). Funksiyaning prototipi quyidagicha:

```
#include <ctime>
time_t time(time_t *Time);
```

Funksiyani ikki xil usul bilan chaqirish mumkin: parametr sifatida nolinchi ko'rsatkich yoki qaytarilayotgan qiymat yoziladigan o'zgaruvchi manzilini uzatib. Masalan:

```
time_t t1 = time(0); // Nolinchi ko'rsatkich beramiz
cout << t1 << endl; // 1483882506
time_t t2;
time (&t2); // O'zgaruvchining manzilini beramiz
cout << t2 << endl; // 1483882506
```

time() funksiyasi o'rniga _time32() va _time64() funksiyalaridan foydalanish mumkin. Qavs oldidagi son bitlar sonini anglatadi.

Funksiyalarning prototiplarii:

```
time32_t _time32(time32_t *Time) ;
time64_t _time64(time64_t *Time) ;
```

Qo'llashga misol:

```
time32_t t1;
```

```
_time32 (&t1); cout << t1 << endl; // 1483882506
time64_t t2;
_time64 (&t2); cout << t2 << endl; // 1483882506
```

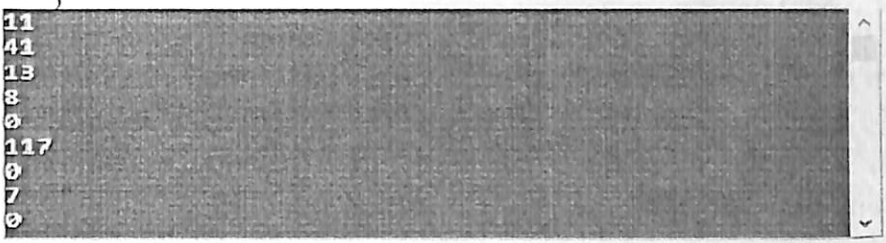
gmtime() – universal sanali (UTC) tm strukturaga ko'rsatkich yoki xatolikka nolinchi ko'rsatkich qaytaradi. Parametr sifatida davr boshidan boshlab (*01.01.1970-y.*) o'tgan sekundlar sonini olgan o'zgaruvchining manzili ko'rsatiladi. Joriy sanani parametr sifatida olish uchun time() funksiyasining bajarilishini uzatish zarur. Funksiya prototipi quyidagicha:

```
#include <ctime>
struct tm *gmtime(const time_t *Time);
```

Funksiyadan foydalanishga misol:

```
#include "stdafx.h"
using namespace std;
using namespace System;
#include <iostream>
#include <ctime>
int main() {
tm *ptm = 0;
time_t t = time(0) ;
ptm = gmtime(&t);
if(!ptm) {
cout << "Xato" << endl;
exit(1);
}
cout << ptm->tm_sec << endl; // 11
cout << ptm->tm_min << endl; // 41
cout << ptm->tm_hour << endl; // 13
cout << ptm->tm_mday << endl; // 8
cout << ptm->tm_mon << endl; // 0 (yanvar)
cout << ptm->tm_year << endl; // 117(2017-y.)
cout << ptm->tm_wday << endl; // 0 (yakshanba)
cout << ptm->tm_yday << endl; // 7
```

```
cout << ptm->tm_isdst << endl; // 0
cin.get();
return 0;
}
```



VC++ da gmtime() funksiyasi o'rniga _gmtime32() va _gmtime64() funksiyalaridan foydalanish mumkin. Qavs oldidagi son bitlar sonini aniqlaydi. Funksiyaning prototipi quyidagicha ko'rinishda:

```
struct tm *_gmtime32(const time32_t *Time);
struct tm *_gmtime64(const time64_t *Time);
```

localtime() funksiyasi lokal vaqt bilan tm strukturasiga ko'rsatkich yoki xatolik yuzaga kelganda nol ko'rsatkich qaytaradi. Parametr sifatida davr boshidan boshlab sekundlar sonini olgan o'zgaruvchi manzili ko'rsatiladi. Parametr sifatida joriy sanani olish uchun time() funksiyasining bajarilish natijasini berish lozim. Funksiyaning umumiy ko'rinishi:

```
#include <ctime>
struct tm * localtime(const time_t *Time);
```

Funksiyadan foydalanishla misol:

```
tm *ptm = 0;
time_t t = time(0);
ptm = localtime(&t);
if(!ptm) {
cout << "Xato" << endl;
exit(1);
}
cout << ptm -> tm_sec << endl;
```

```
cout << ptm -> tm_min << endl;
cout << ptm -> tm_hour << endl;
cout << ptm -> tm_mday << endl;
cout << ptm -> tm_mon << endl;
cout << ptm -> tm_year << endl;
cout << ptm -> tm_wday << endl;
cout << ptm -> tm_yday << endl;
cout << ptm -> tm_isdst << endl;
```

localtime() funksiyasi o'rniga _localtime32() va _localtime64() funksiyalaridan ham foydalanish mumkin. Qavs oldidagi son bitlar sonini ko'rsatadi. Funksiyalarning prototiplari quyidagicha:

```
struct tm *_localtime32(const time32_t *Time) ;
struct tm *_localtime64(const time64_t *Time);
```

Bu funksiyalardan foydalanilganda ogohlantiruvchi xabar chiqariladi ("warning C4996"), shu sababli ularning o'rniga mos ravishda quyidagi funksiyalardan foydalanish tavsiya qilinadi:

```
#include <ctime>
errno_t localtime_s(struct tm *Tm, const time_t *Time);
errno_t _localtime32_s(struct tm *Tm, const time32_t
*Time);
errno_t _localtime64_s(struct tm *Tm, const time64_t
*Time);
```

Agar xatolik yuzaga kelmasa, funksiya '0' qiymat qaytaradi. Xatolik mavjud bo'lganda EINVAL makrosining qiymati (22) qaytariladi va errno o'zgaruvchisi EINVALga teng deb o'rnatiladi. localtime_s() funksiyasidan foydalanishga misol ko'ramiz.

```
tm ptm;
time_t t = time(0);
errno_t err = localtime_s(&ptm, &t);
if(err) {
 cout << "Xato" << endl;
```

```
exit(1);
}
cout << ptm.tm_mday << endl; // 8
cout << ptm.tm_mon << endl; // 0 (yanvar)
cout << ptm.tm_year << endl; // 117 (2017-y.)
```

mktime() – davrsdan boshlab o‘tgan sekundlar sonini qaytaradi. Parametr sifatida tm strukturaga ko‘rsatkich qaytaradi. Xatolik yuzaga kelganda –1 qiymat qaytaradi. Funksiyaning prototipi:

```
#include <ctime>
time_t mktime(struct tm *Tm) ;
```

Funksiyadan foydalanishga doir misol:

```
tm ptm;
time_t t1 = time(0), t2 = 0;
errno_t err = localtime_s(&ptm, &t1);
if(err) {
 cout << "Xato" << endl;
 exit(1);
}
t2 = mktime(&ptm);
cout << t1 << endl; // 1285636288
cout << t2 << endl; // 1285636288
```

mktime() funksiyasi o‘rniga _mktime32() va _mktime64() funksiya-laridan foydalanish mumkin. sonlar bitlar sonini anglatadi:

```
time32_t _mktime32(struct tm *Tm);
time64_t _mktime64(struct tm *Tm);
```

difftime() – ikkita vaqt oralig‘idagi farqni qaytaradi (Time1-Time2). Xatolik yuzaga kelganda ‘0’ qiymat qaytaradi va errno o‘zgaruvchisi EINVAL ga teng deb o‘rnatiladi. Funksiyamimg umumiy ko‘rinishi:

```
#include <ctime>
double difftime(time_t Time1, time_t Time2);
```

Misol:

```
time_t t1 = time(0), t2 = 1233368623;
double result = difftime(t1, t2);
cout << result << endl;
```

difftime() funksiyasi o'ringa _difftime32() va _difftime64() funksiyalaridan foydalanish mumkin. Ochiluvchi qavs oldidagi son bitlar sonini anglatadi. Funksiyalarning prototiplari:

```
double _difftime32(time32_t Time1, time32_t Time2);
double _difftime64(time64_t Time1, time64_t Time2);
```

Quyidagi misolda joriy vaqt va sanani shunday chiqaramiz-ki, bunda hafta kuni va oy o'zbek tilida yozilsin.

```
#include <iostream>
#include <clocale>
#include <ctime>
#include <iomanip> // setfill() va setw() uchun
int main() {
setlocale(LC_ALL, "Russian_Russia.1251");
tm ptm;
time_t t = time(0);
char d[][25] = {"yakshanba", "dushanba", "seshanba",
"chorshanba", "payshanba", "juma", "shanba"};
char m[][25] = {"yanvar", "fevral", "mart", "aprel",
"may", "iyun", "iyul", "avgust",
"sentyabr", "oktyabr", "noyabr", "dekabr"};
errno_t err = localtime_s(&ptm, &t); // Joriy vaqtni olish
if(err) {
cout << "Xato" << endl;
exit(1);
}
 cout << "Bugun: " << endl << d[ptm.tm_wday]
<< " " << ptm.tm_mday << " " << m[ptm.tm_mon]
<< " " << (ptm.tm_year + 1900);
cout << setfill('0') << " " << setw(2) << ptm.tm_hour
```

```
<< ":" << setw(2) << ptm.tm_min << ":" << setw(2)
<< ptm.tm_sec << endl << setw(2) << ptm.tm_mday
<< "." << setw(2) << (ptm.tm_mon+1) << "."
<< (ptm.tm_year + 1900) << endl;
    cin.get();
    return 0;
    }
```

```
Bugun:
yakshanba 8 yanvar 2017 19:03:22
08.01.2017
```

Keltirib o'tilgan misolda iomanip sarlavha faylida aniqlangan setfill() va setw() monipulyatorlaridan foydalanildi. setfill() monipulyatori belgi-to'ldirgichni, setw() monipulyatori maydon kengligini ko'rsatish uchun tayinlangan. Agar bu monipulyatorlardan foydalanilmasa, vaqt "16:01:05" o'rniga "16:1:5" ko'rinishda bo'ladi.

10.2. Sana va vaqtni formatlash

Quyidagi funksiyalar sana va vaqtni formatli chiqarishni ta'minlaydi:

asctime() – maxsus fotmatdagi C-satrga ko'rsatkich yoki xatolik yuzaga kelganda nol ko'rsatkich qaytaradi. Satr oxiriga satrni ko'chirish (\n) va nol-belgi (\0) qo'yiladi. Funksiyaning prototipi:

```
#include <ctime>
char *asctime(const struct tm *Tm);
```

Funksiyadan foydalanishga misol:

```
tm ptm;
char *p = 0;
time_t t = time(0);
errno_t err = localtime_s(&ptm, &t);
if(err) {
cout << "Xato" << endl;
```

```
exit(1);
}
p = asctime(&ptm);
if(!p) {
cout << "Xato" << endl;
exit(1);
}
cout << p << endl;
```

```
Mon Jan 09 20:34:27 2017
```

asctime() funksiyasi ogohlantiruvchi xabar ("warning C4996") chiqaganligi uchun uning o'rniga asctime_s() funksiyasidan foydalanish tavsiya qilinadui. Funksiyaning prototipi:

```
#include <ctime>
errno_t asctime_s(char *Buf, size_t SizeInBytes,
const struct tm *Tm);
```

Birinchi parametrda ko'rsatkich satrga uzatiladi, ikkinchi parametrda satrning maksimal uzunligi, uchunchi parametrda esa tm strukturaga ko'rsatkich. Agar xatolik bo'lmasa, funksiya nol qaytaradi. Xatolik yuzaga kelganda EINVAL makrosining qiymati (qiymati: 22) qaytariladi. Funksiyadan foydalanishga misol:

```
tm ptm;
const short SIZE = 80;
char str[SIZE] = {0};
time_t t = time(0);
errno_t err = localtime_s(&ptm, &t);
if(err) {
 cout << "Xato" << endl;
 exit(1);
}
err = asctime_s(str, SIZE, &ptm);
if(err) {
 cout << "Xato" << endl;
```

```
exit(1);
}
cout << str << endl;
```

```
Tue Jan 17 20:09:46 2017
```

ctime() – asctime() funksiyasi bilan bir xil, ammo parametr sifatida 1970-yil 1-yanvardan boshlab o'tgan sekundlar sonini oladi. Funksiya prototipi quyidagicha ko'rinishga ega:

```
#include <ctime>
char *ctime(const time_t *Time);
```

Funksiyadan foydalanishga misol:

```
char *p = 0;
time_t t = time(0);
p = ctime(&t);
if(!p) {
 cout << "Xato" << endl;
 exit(1);
}
cout << p << endl;
```

```
Mon Jan 09 22:57:04 2017
```

Funksiya ogohlantiruvchi xabar ("warning C4996") chiqariladi, shu sababli uning o'rniga ctime_s() funksiyasidan foydalanish tavsiya qilinadi. Funksiya prototipi:

```
#include <ctime>
errno_t ctime_s(char *Bufer, size_t SizeInBytes,
const time_t *Time);
```

Xatoliklar bo'lmasa, funksiya "0" qaytaradi. Xatoliklar mavjud bo'lganda EINVAL makrosi qiymati qaytariladi. Funksiyadan foydalanishga misol:

```
const short SIZE = 80;
char str[SIZE] = {0};
time_t t = time(0);
errno_t err = ctime_s(str, SIZE, &t);
```

```
if(err) {
 cout << "Xato" << endl;
 exit(1);
}
cout << str << endl; // Natija: Mon Jan 09 22:57:04
2017\n\0
```

strftime() – sananing satrli ko'rinishini Buf C-satrga mos holda formatli yozadi. Funksiya prototipi:

```
#include <ctime>
size_t strftime(char *Buf, size_t SizeInBytes,
const char *Format, const struct tm *Tm);
```

Birinchi parametrda, funksiyaning bajarilishi yoziladigan, belgili massivga ko'rsatkich uzatiladi. SizeInBytes parametrida belgili massivning maksimal uzunligi beriladi. Format parametrida maxsus format satri ko'rsatiladi, so'nggi parametrda taqdim etilgan vaqt bilan tm strukturaga ko'rsatkich uzatiladi. Funksiya yozilgan belgilar sonini qaytaradi. Xatolik yuzaga kelganda errno o'zgaruvchisi EINVAL makrosiga teng deb o'rnatiladi.

strftime() funksiyasidan foydalanishga misol tariqasida joriy sana va vaqtni chiqarish dasturini ko'raylik.

```
#include <iostream>
#include <clocale>
#include <ctime>
int main() {
 setlocale(LC_ALL, "Russian_Russia.1251");
 tm ptm;
 const short SIZE = 100;
 char str[SIZE] = {0};
 time_t t = time(0);
 int err = localtime_s(&ptm, &t);
 if(err) {
 cout << "Xato" << endl; exit(1);
 }
```

```
    err = strftime(str, SIZE,"Сегодня:\n%A %d %b %Y %H:
%M:%S\n%d.%m.%Y", &ptm);
    if(!err) {
    cout << "Xato" << endl;
    exit(1);
    }
    cout << str << endl;
    cin.get();
    return 0;
    }
```

Dasturning bajarilishini to'xtatib turish. sleep() Win API-funksiyasi dasturning bajarilish jarayonini ko'rsatilgan vaqt davomida to'xtatib turush imkonini beradi. Vaqt tugaganidan so'ng, dastur ishini davom ettiradi. Funksiyaning prototipi:

```
#include <Windows.h>
void Sleep(DWORD dwMilliseconds) ;
```

Funksiyaning dvMilliseconds parametrida dasturning bajarilishi qancha vaqtga to'xtab turishi millisekundlarda ko'rsatiladi. DWORD berilganlar tipi quyidagicha aniqlangan:

```
typedef unsigned long DWORD;
```

Misol tariqasida 1 dan 10 gacha sonlarni chiqaramiz. Sonlarni chiqarish oralig'ida bir sekundga "to'xtaymiz". Dastur listinggi quyidagicha ko'rinishga ega bo'ladi:

```
#include <iostream>
#include Windows.h>
int main() {
 for(int i = 1; i <= 10; ++i) {
 cout << "\r... " << i << "%";
 Sleep(1000); // 1 sekundga "to'xtaydi"
 }
 cout << "\rTamom" << endl;
```

```
cin.get() ;
return 0;
}
```

10.3. Kod qismining bajarilish vaqtini o'lchash

Ba'zi hollarda kod parchalarining (masalan dasturni optimallashtirish maqsadida) bajarilish vaqtini o'lchashga to'g'ri keladi. Buning uchun clock() funksiysidan foydalaniladi. Funksiyaning prototipi quyidagicha:

#include <ctime> clock_t clock(void);

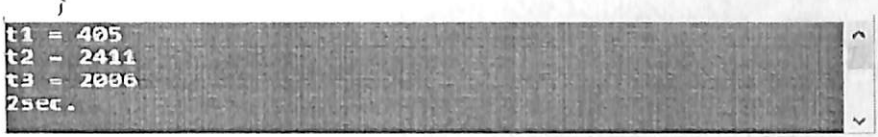
Bu funksiya funksiya chaqirilgunga qadar dastur bajarilishining taxminiy qiymatini qaytaradi. Agar vaqtni olish imkoniyati bo'lmasa, funksiya "-1" qiymat qaytaradi. Dastur qismining (dasturning) bajarilish vaqtini o'lchash uchun funksiya parchadan oldin chaqilishi va natijani saqlashi lozim. So'ngra funksiya parchadan keyin chaqiriladi va ikki qiymat orasidagi farq hisoblanadi. Qiymatni sekundlarda olish uchun natijani CLOCKS_PER_SEC makrosi qiymatiga bo'lish kerak. Makrosning aniqlanishi quyidagicha ko'rinishga ega:

#define CLOCKS_PER_SEC 1000

Misol uchun, Sleep() WinAPI-funksiyasi yordamida kod qismining bajarilishini imitatsiya qilamiz va bajarilish vaqtini o'lchaymiz.

```
#include <iostream>
#include <ctime>
#include <Windows.h>
int main() {
 clock_t t1, t2, t3;
 t1 = clock(); // 1-metka
 cout << "t1 = " << t1 << endl;
 Sleep(2000) ; // Kod fragmentini imitatsiyalash
 t2 = clock(); // 2-metka
```

```
    cout << "t2 = " << t2 << endl;
    t3 = t2 – t1; // Metkalar orasidagi farq
    cout << "t3 = " << t3 << endl;
    cout << (t3 / CLOCKS_PER_SEC) << "sec." << endl;
    cin.get();
    return 0;
    }
```



Nazorat savollari

1. Kompilyatsiya jarayoni nima?
2. Kompilyatsiya jarayonining necha bosqichdan tashkil topadi?
3. Qanday belgilar bilan tugagan barcha belgilar ketma-ketligi izoh hisoblanadi?
4. C++ tilining kalit so'zlariga qaysilar kiradi?
5. Protsessor registrlarini belgilash uchun qaysi so'zlar ishlatiladi?
6. Kiritilgan qiymatni o'zgaruvchi turiga mos kelishini qanday tekshirish mumkin?
7. Identifikator nima?
8. C++ ko'rsatmalari qanday belgi bilan tugallanishi zarur?
9. Tilning ma'nosini beruvchi qoidalar to'plami qanday nomlanadi?
10. int turidagi son o'zgaruvchisi uchun dastur qismi ko'rsatib bering.
11. Agar dasturda sintaktik xatolar bo'lsa, kompilyator bu haqida xabar beradimi?
12. Arifmetik operatorlar bajarilish ketma-ketligi qoidasi qanday qoida?
13. Figurali qavslar nima uchun ishlatiladi?
14. Vergul (",") odatda nima uchun ishlatiladi?
15. "double" kalit so'zi nima uchun ishlatiladi?
16. Haqiqiy son turi nima?
17. Berilganlarning strukturalashgan turlari qanday?
18. Butun son turlari nima?
19. Sanab o'tiluvchi tur nima?
20. Haqiqiy sonlar qanday kalit so'zi bilan e'lon qilinadi?
21. O'nlik fiksirlangan nuqtali format deganda nimani tushuniladi?
22. Eksponensial shaklda haqiqiy o'zgarmas necha qismdan iborat bo'ladi va ularga misol ko'rsating.

23. Harf belgilar qanday registrlarda berilishi mumkin?

24. Kompilyator nimaga qarab unga mos turni belgilaydi?

25. Aralash ifodada qanday hisoblanadi?

26. Qaysi operator yordamida oshkor ravishda bir turni boshqa turga keltirish mumkin?

27. cast operatori yordamida belgilar boshqa turga keltirish mumkinmi?

28. C++ tilida tuzilgan dasturning asosiy maqsadi nima?

29. O'zgaruvchi nima?

30. O'zgaruvchilarga ifoda qanday belgi orqali yuklanadi?

31. C++ tilida num = num +2; ko'rinishidagi ifoda nimani bildiradi?

32. Kod qismidagi o'zgaruvchilarning kompilyator uchun qanday ketma-ketlikda qiymat olishlarini jadvalini yozing.

33. C++ tilida bir turni boshqa turga keltirishning qanday yo'llari mavjud?

34. Inkrement va dekrement amallari nima?

35. Prefiks yoki postfiks amal tushunchasi qanday ifodalarda o'rinli?

36. Qo'yilgan masalani yechishda biror holat ro'y bergan yoki yo'qligini ifodalash uchun 0 va 1 qiymat qabul qiluvchi nimalardan foydalaniladi?

37. C++ tilida bayt razryadlari ustida mantiqiy amallar majmuasi jadvalini ko'rsating.

38. Baytdagi bitlar qiymatini chapga yoki o'ngga surish uchun, mos ravishda qaysi amallari qo'llaniladi?

39. Taqqoslash amali qanday amal bo'lib, u qanday ko'rinishga ega?

40. Taqqoslash amallarining natijasi - taqqoslash o'rinli bo'lsa yoki o'rinli bo'lmasa qanday qiymat bo'ladi?

41. Ifodalar qiymatini hisoblashda nima hisobga olinadi?

42. O'qish oqimi nima?

43. Input stream nima?

44. Output stream nima?

45. include operatori qanday vazifani bajaradi?

46. cout operatori qanday vazifani bajaradi?

47. cin operatori qanday vazifani bajaradi?

48. C++ dastur sarlavhasida qaysi fayldan foydalanish kerak?

49. iostream fayli nechta oqimdan tashkil topgan?

50. Qiymat dastur orqali o'qib olinganida nimalar qiymatlarning ajratuvchisi sifatida qabul qilinadi?

51. Berilganlar oqimidan faqat kerakli qismini kiritish kerak bo'lsa, unda kiritish oqimining qaysi funksiyasidan foydalanish kerak?

52. Mantiqiy qo'shish operatori nechta ifoda orqali hisoblanadi?

53. Mantiqiy inkor operatori tekshirilayotgan ifoda yolg'on bo'lsa qanday qiymat qaytaradi?

54. if operatori qanday funksiyani bajaradi?

55. else operatori qanday funksiyani bajaradi?

56. C++ tilining qurilmalari operatorlarni blok ko'rinishida tashkil qilishga imkon beradimi? Buni tushuntirib bering.

57. Blok – nima?

58. Shart operatorida e'lon qilish operatorlarini ishlatish mumkinmi?

59. Agar tekshirilayotgan shart nisbatan sodda bo'lsa nima ishlatish mumkin?

60. switch tarmoqlanish operatori nima?

61. break va default kalit so'zlari nimauchun ishlatiladi?

62. switch operatorida e'lon operatorlari ham uchrashi mumkinmi?

63. switch operatori bajarilishida "sakrab o'tish" holatlari bo'lishi hisobiga blok ichidagi ayrim e'lonlar bajarilmasligi va buning oqibatida dastur ishida xatolik ro'y berishi mumkinmi?

64. switch operatori nima uchun ishlatiladi?

65. Sanab o‘tiluvchi turlar va shu turdagi o‘zgaruvchilarga misol keltiring.

66. Mantiqiy operatorlarga nimalar kiradi?

67. <operand1><taqqoslash amali>< operand2> quyidagi amal nimani anglatadi?

68. “&&” “||” “!” operatorlari nimani anglatadi?

69. (x==3) && (y==5) agar x va y qiymatlari har hil bo‘lsa qanday qiymat qaytaradi?

70. Mantiqiy ko‘paytirish operatori qanday belgi orqali belgilanadi?

71. Mantiqiy qo‘shish operatori qanday belgi orqali belgilanadi?

72. Beshta sonning o‘rta arifmetigi qanday topiladi?

73. Cheksiz takrorlash uchun takrorlashni davom ettirish sharti?

74. Takrorlash operatorida ham bloklardan foydalanish mumkinmi?

75. C++ tilining qurilmalari operatorlarni blok ko‘rinishida tashkil kilishga imkon beradimi?

76. Agar <ifoda> rost qiymatli o‘zgarmas ifoda bo‘lsa, takrorlash qanday buladi?

77. Takrorlash operatorlarining bajarilishida qanday holatlar yuzaga kelishi mumkin?

78. Takrorlash operatori ichma-ich joylashgan bo‘lishi mumkunmi?

79. do-while takrorlash operatori qanday vazifani bajaradi?

80. while takrorlash shartini oldindan tekshiruvchi takrorlash operatori hisoblanadimi?

81. continue operatori qanday vazifani bajaradi?

82. break operatori qanday vazifani bajaradi?

83. while operatori qanday vazifani bajaradi?

84. for operatori qanday vazifani bajaradi?

85. Ayrim hollarda, goto operatorining «sakrab o‘tishi» hisobiga xatoliklar yuzaga kelishi mumkunmi?

86. Takrorlash operatorida qavs ichidagi ifodalar bo'lmasligi mumkin, lekin sintaksis ';' bo'lmasligiga ruxsat beriladimi?

87. Massiv deb nimaga aytiladi?

88. Massiv indeksi sifatida qanday son ishlatiladi?

89. Dasturda ishlatiladigan har bir konkret massiv qanday nomga ega?

90. Massiv elementiga murojaat qilish qanday amalga oshiriladi?

91. C++tilida massivlar elementining turiga cheklovlar kuyiladimi?

92. Ikki o'lchamli massivning sintaksisi qanday ko'rinishda bo'ladi?

93. So'zlar massivlari initsializatsiya qilinganda so'zlar soni ko'rsatilmasligi mumkin. Bu holda so'zlar soni qanday aniqlanadi?

94. Massiv – bu?

95. Misolda massiv elementlar soni keltirilmagan bulsa massiv elementlar soni qanday aniqlanadi?

96. Funksiya bu...?

97. Funksiyalar modullar deb ham atalishi mumkunmi?

98. C++ tilida funksiya chaqirilganda ayrim argumentlarni tushirib qoldirish mumkunmi va bunga qanday erishish mumkun?

99. Lokal o'zgaruvchilar o'zlari e'lon qilingan funksiya yoki blok chegarasida ko'rinish sohasiga ega buladimi?

100. Funksiyalar qanday turlarga bo'linadi?

101. Funksiyalar qanday ko'rinishda bo'ladi?

102. Kompilyator ishlashi natijasida har bir funksiya qanday ko'rinishida bo'ladi?

103. Ayrim algoritmlar berilganlarning qanday turdagi qiymatlari uchun qo'llanishi mumkin?

104. Qayta yuklanuvchi funksiyalardan foydalanishda qanday qoidalarga rioya qilish kerak?

105. Inline operatori qanday funksiyani bajaradi

106. Qiymatni hisoblash uchun funksiyaning «oldingi qiymati» ma`lum bo‘lishi kerakmi?

107. Rekursiya deb nimaga aytiladi?

108. Agar faktorial funksiyasiga n>0 qiymat berilsa, qanday holat ro‘y beradi?

109. Rekursiv funksiyalarni to‘g‘ri amal qilishi uchun qanday chaqirishlarning to‘xtash sharti bo‘lishi kerak?

110. Har bir rekursiv murojaat qo‘shimcha xotira talab qiladimi?

111. Rekursiya qanday to‘xtatiladi?

112. Har bir rekursiv formula nechta ifodaga ega bo‘lishi kerak?

113. Sanab o‘tiluvchi turlar nima maqsadda ishlatiladi?

114. Sanab o‘tiluvchi turni aniqlash qanday qismlardan iborat?

115. Sanab o‘tiluvchi turlar ustida qanday amallar bajarib bo‘lmaydi?

116. typedef kalit so‘zi yordamida yangi tur hosil qilishga misol keltiring.

117. Nomlar fazosi nima uchun xizmat qiladi?

118. Nomlar fazosi o‘zgaruvchilariga qanday murojaat qilinadi?

119. Statik o‘zgaruvchilar nima uchun xizmat qiladi?

120. Tashqi o‘zgaruvchilar nima uchun xizmat qiladi?

121. Lokal va global o‘zgaruvchilarning bir-biridan farqi nimada?

122. Qiymatlari adres bo‘lgan o‘zgaruvchilarga nima deyiladi?

123. Funksiyaga ko‘rsatkichning yozilish sintaksisi qanday bo‘ladi?

124. Obyektga ko‘rsatkich e’loni qanday bo‘ladi?

125. void ko‘rsatkichining muhim afzalliklari nimalardan iborat?

126. Dinamik o‘zgaruvchilar deb nimaga aytiladi?

127. Ko‘rsatkichga boshlang‘ich qiymat berish qay tarzda amalga oshiriladi?

128. Ko‘rsatkich ustida qanday amallar bajarilishi mumkin?

129. new operatori natija sifatida nimani qaytaradi?

130. Dinamik xotirada new amali bilan joy ajratish?

131. Kerak bo‘lmagan xotirani qaysi operator yordamida bo‘shatish mumkin?

132. Dinamik massiv bilan statik massivning farqini aytib bering.

133. Dinamik massiv e’lementlari miqdorini qanday ko‘rsatish mumkin?

134. O‘zgaruvchi parametrli funksiyalar qanday e’lon qilinadi?

135. Dinamik massiv elementlari miqdorini qanday ko‘rsatish mumkin?

136. Funksiyada bir o‘lchamli statik massiv qanday ishlatiladi?

137. Funksiyada bir o‘lchamli dinamik massiv qanday ishlatiladi?

138. Funksiyada ko‘p o‘lchamli statik massiv qanday ishlatiladi?

139. Funksiyada ko‘p o‘lchamli dinamik massiv qanday ishlatiladi?

140. Struktura deb nimaga aytiladi?

141. Birlashma deb nimaga aytiladi?

142. Birlashma va strukturaning farqi nimada?

143. Struktura maydonlari qanday turdarda bo‘lishi mumkin?

144. Strukturani funksiya argumenti sifatida ishlatishga misol keltiring.

145. Strukturalar massivi qanday e’lon qilinadi?

146. Struktura maydonlariga qanday murojaat qilish mumkin?

147. Strukturaga ko‘rsatkich qanday ishlatiladi?
148. Ichma-ich strukturalar qanday ishlatiladi?
149. Lokal va global o‘zgaruvchilarning farqi nimada?
150. :: amali nima uchun xizmat qiladi?
151. define direktivasi nima uchun xizmat qiladi?
152. Qanday o‘qish oqimlarini bilasiz?
153. Qanday yozish oqimlarini bilasiz?
154. Belgilarni o‘qish uchun qaysi funksiyalar ishlatiladi?
155. Belgilarni yozish uchun qaysi funksiyalar ishlatiladi?
156. Satrlarni o‘qish uchun qaysi funksiyalar ishlatiladi?
157. Satrlarni yozish uchun qaysi funksiyalar ishlatiladi?
158. Satr qismini izlash uchun qanday funksiyadan foydalanish mumkin?
159. Satr qismini qanday o‘chirish mumkin?
160. Satrlarni ulash uchun nima qilish kerak?

Glossariy

Termin	O'zbek tilidagi sharhi
adres (manzil)	o'zgaruvchi xotirada joylashadigan adres
amal qilish sohasi	o'zgaruvchini ishlatish mumkin bo'lgan dastur sohasi
argument	funksiyaga parametriga jo'natiladigan qiymat
bayt	kompyuter xotirasi o'lchov birligi
berilganlar	dastur ishlashi uchun kerakli qiymatlar
binar amal	ikkita operand ustida bajariluvchi amal
birlashma	maydonlariga umumiy joy ajratiladigan tuzilma
bit	eng kichik o'lchov birligi
break	takrorlashni to'xtatish operatori
case	konstantalar bilan tekshirish operatori
char	belgi ko'rinishidagi berilganlarning turi
cheksiz takrorlash	takrorlashni to'xtatish shartining mavjud emasligi
cin	ekrandan kiritish oqimi
continue	takrorlash keyingi qadamiga o'tkazish operatori
cout	ekranga chiqarish oqimi
define	makrosni e'lon qiluvchi direktiva

dekrement	o‘zgaruvchining qiymatini bittaga kamaytirish
delete	xotiradan ajratilgan joyni tozalash operatori
double	haqiqiy son ko‘rinishidagi berilganlarning turi
do-while	sharti keyin tekshiriladigan takrorlash operatori
else	shart yolg‘onligini aniqlovchi operator
enum	sanab o‘tiluvchi tur
EOF	#define EOF(-1) ko‘rinishida aniqlangan makros
for	takrorlash qadami bilan beriladigan takrorlash operatori
fstream	fayl oqimi
funksiya	dastur alohida bo‘lagi, asosiy qism tomonidan chaqirib ishlatiladi
identifikator	katta va kichik lotin harflari, raqamlar va tag chiziq (‘_’) belgilaridan tashkil topgan va raqamdan boshlanmaydigan belgilar ketma-ketligi
if	shart operatori
ifstream	o‘qish fayli oqimi
include	preprotsessor direktivasi, kutubxona fayllarni dasturga ulash uchun ishlatiladi

inkrement	o‘zgaruvchining qiymatini bittaga oshirish
int	butun son ko‘rinishidagi berilganlarning turi
kengaytma	fayllarning turli dasturlarga tegishliligini aniqlovchi fayl ko‘rinishining qismi
ko‘rsatkich	qiymatlari adres bo‘lgan o‘zgaruvchilar
kompilyatsiya	bajariluvchi fayl hosil bo‘lish jarayoni
konstanta	dastur davomida qiymati o‘zgarmaydigan berilgan
kutubxona	dasturga include direktivasi yordamida qo‘shiladigan fayllar
leksema	tilning ajralmaydigan qismlari
namespace	nomlar fazosini e’lonini aniqlovchi kalit so‘z
new	xotiradan yangi joy ajratish operatori
NULL	mavjud bo‘lmagan qiymat
o‘zgarmas	dastur ishlashi davomida qiymatini o‘zgartirmaydigan berilgan
o‘zgaruvchi	berilganlarni saqlab turish uchun ishlatiluvchi til birligi
ofstream	Yozish fayli oqimi
parametr	funksiya ishlashi uchun kerak berilganlar

postfiks	operatorning o‘zgaruvchidan keyin joylashgan ko‘rinishi
prefiks	operatorning o‘zgaruvchidan oldin joylashgan ko‘rinishi
razryad	bitlardan (0 yoki 1) tashkil topgan indikator
sarlavha fayli	funksiyalar e’loni yozilgan fayl
semantika	tilning ma’nosini beruvchi qoidalar to‘plami
setw	o‘zgaruvchining belgi bilan to‘ldirib chiqarish
sintaktik qoidalar	grammatik qoidalarga o‘xshash qoidalar to‘plami
sizeof	o‘zgaruvchi turining xotiradagi hajmini aniqlash
struktura	bir yoki har xil turdagi berilganlarni jamlanmasi
switch	bir nechta konstanta bilan tekshirish operatori
typedef	turlarni yangi nom bilan ishlatish imkonini beradi
unar amal	bitta operand ustida bajariluvchi amal
using	nomlar fazosini dasturga ulash uchun kalit so‘z
while	sharti oldin tekshiriladigan takrorlash operatori

Adabiyotlar ro'yxati

1. Madraximov Sh.F., Ikramov A.M., Babajanov M.R. C++ tilida programmalash bo'yicha masalalar to'plami. O'quv qo'llanma // Toshkent, O'zMU, "Universitet" nashriyoti, 2014.

2. T.A.Maxarov, Q.T.Maxarov. Visual Studio muhitida dasturlash asoslari (uslubiy qo'llanma). Toshkent, O'zMU. 2017.

3. Xaldjigitov A.A., Madraximov Sh.F., Adamboev U.E. Informatika va programmalsh. O'quv qo'llanma. O'zMU, 2005.

4. Xoldjigitov A., Adambayev U.E. Programmalash asoslari. Toshkent. -2005.

5. Ғуломов С.С., Бегалов Б.А., Зайналов Н.Р., Дадабаева Р.А., Давронов А.Э.. Дастурлаш технологиялари. Олий ўкув юртлари учун ўкув кўлланма. - Тошкент.: ТДИУ, 2006.

6. Мадрахимов Ш.Ф., Гайназаров С.М. С++ тилида программалаш асослари// Тошкент, ЎзМУ, 2009.

7. Хашимова Д.П., Насридинова Ш.Т. Технологии программирования. Учебное пособие. – Ташкент: ТГЭУ, 2012.

8. Bjarne Stroustrup. Programming: Principles and Practice using C++ (Second Edition)" Addison-Wesley, 2014.

9. Bjarne Stroustrup. The C++ Programming Language (4th Edition). Addison-Wesley, 2013.

10. Brain Overland. C++ Withour Fear. New Jersey, 2005

11. D.S.Malik. C++ Programming: From Problem Analysis to Program Design, Fifth Edition. Course Technology. Printed in the USA. 2011.

12. Ivor Horton. Beginning Visual C++ 2005. Wiley Publishing, 2005.

13. Scheinerman Edwant C++ for Mathematicifns. AnIntroduction for Students and Professionals. Chapman&Hall/CRC,Taylor&Francis Group, LLC, Boca Raton, London, New York, 2006.

14. Walter Savitch. Absolute C++, 5th edition. Addison-Wesley/Pearson, 2012.

15. Walter Savitch. Problem Solving with C++, 9th edition. Addison-Wesley/Pearson, 2015.

16. Т.А.Павловская, Ю.А.Щупак С и С++. Структурное программирование: Практикум. СПб. Питер 2003.

17. А.Я.Архангельский. Программирование в С++ Builder. 7-изд. –М: ООО «Бином Пресс», 2010.

18. Гагарина Л.Г., Кокорева Е.В., Виснадул Б.Д. Технология разработки программного обеспечения: учебное пособие /под ред. Л.Г. Гагариной. – М.: ИД «ФОРУМ»: ИНФРА-М, 2008.

19. Герберд Шилдт. С++: базовый курс, 3-е издание. Перевод с английского. –М: Изд.дом "Вильямс", 2010.

20. Глушаков С.В., Коваль А.В., Смирнов С.В. Язык программирования С++: Учебный курс.- Харьков: Фолио; М.: ООО «Издательство АСТ», 2001.

21. Голицына О.Л., Партыка Т.Л., Попов И.И. Языки программи-рования. Учебное пособие. – М.: ФОРУМ: ИНФРА-М, 2008.

22. Культин Н.Б. С++ Builder в задачах и примерах.- СПб.: БХВ-Петербург, 2005.

23. О.Е.Степаненко. Visual C++.NET. Классика программирования.: - М: Научная книга, К.; Букинист, 2010.

24. Павловская Т.А. С++. Программирование на языке высокого уровня – СПб.: Питер. 2005.

25. Пахомов Б.И. С/С++ и MS Visual Studio 2010 для начинающих. СПб. БХВ-Петербург. -2011.

26. Стивен Прата. Язык программирования С++. Лекции и упражнения. Учебник. С-Пб. ООО «ДиаСофтЮП», 2005

27. Фаронов В.В. Delphi. Программирование на языке высокого уровня: Учебник. – СПб.: Питер, 2009.

28. Хортон, Айвор. Visual C++ 2010: Полный курс. Перевод с английского. М: ООО ИД "Вильямс" -2011.

Internet manbalar

1. http://cplusplus.com – C++ tilida mavjud konstruksiyalar ta'rifi, ishlatish namunalari bilan keltirilgan.

2. http://cppstudio.com – C++ tilida dasturlash bo'yicha namunalar izohlari bilan keltirilgan

3. http://cppstudio.com/

4. http://h-l-l.ru/publ/29-1-0-108

5. http://prog-cpp.ru/cpp/

6. http://rpp.nashaucheba.ru/docs/index-81596.html

7. http://uchebilka.ru/geografiya/132562/index.html

8. http://www.compteacher.ru/programming – dasturlash bo'yicha video-darsliklar mavjud.

9. http://www.iite.ru – YUNESKOning "Ta'limda axborot texnologiyalari" nomli sayti.

10. http://www.intuit.ru – Axborot texnologiyalari internet universiteti, dasturlash bo'yicha yozma va video ma'ruzalar o'qish, test sinovlaridan o'tish va sertifikat olish imkoniyati mavjud.

11. https://msdn.microsoft.com/ru-ru/library/60k1461a.aspx

12. www.ziyonet.uz - O'zbekiston Respublikasi ta'lim portali. Dasturlash bo'yicha referatlar topish mumkin.

Qaydlar uchun

T. A. Maxarov

DASTURLASH ASOSLARI

(Visual Studio)
(o'quv qo'llanma)

Muharrir: *A.Abdujalilov*
Texnik muharrir: *A.Xo'jabekov*
Dizayner: *R.Tashmatov*
Musahhih: *D.O'rinova*
Sahifalovchi: *G.Qurbanbayeva*