O’ZBEKISTON ALOQA VA AXBOROTLASHTIRISH AGENTLIGI

TOSHKENT AXBOROT TEXNOLOGIYALARI UNIVERSITETI

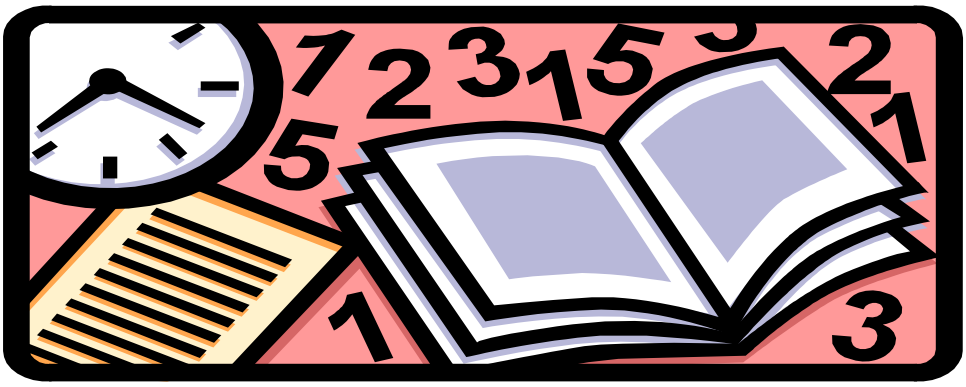
INFORMATIKA KAFEDRASI

INFORMATIKA FANIDAN

C++ TILIDA DASTURLASH

(1-QISM)

USLUBIY QO’LLANMA



TOSHKENT 2005

Hozirgi kunda Respublikamizdagi oliy o'quv yrtlarida «Informatika va axborot texnologiyalari» yo'nalishi talabalariga o'quv rejasiga ko'ra turli xil dasturlash tillarini o'rgatish mo'ljallangan. Ular orasida C++ tili o'zining imkoniyatlari va qo'llanilashi ko'lamiga qarab yuqori o'rinlarda turadi. Mutaxasislarning fikriga ko'ra C++ tili Assambler tiliga eng yaqin bo'lib, tezlik jihatidan Assamblerdan 10% orqada qolar ekan.

Ushbu qo'llanmada C++ algoritmik tili to'g'risida boshlang'ich ma'lumotlar berilgan bo'lib, yangi mavzular orasida ba'zi oddiy dasturlar ham keltirilgan. Bu dasturlarni takomillashtirishni o'quvchilarning o'zlariga xavola qilamiz. Buning uchun ular C++ tili muhitida ishlashni o'rganishlari va qo'shimcha adabiyotlardan foydalanishlari kerak.

Qo'lalanma oson tilida oddiydan murakkabga qarab borish tamoyili asosida yozildi. Ushbu qo'llanma C++ tilida dasturlashni boshlovchilar uchun mo'ljallangan bo'lib, undan barcha bosqichdagi talabalar, magistrantlar, aspirantlar va tilni mustaqil o'rganayotganlar foydalanishlari mumkin. Ushbu qo'llanma Toshkent axborot texnologiyalari universiteti o'quv – uslubiy kengashi qaroriga binoan chop etilmoqda.

KIRISH

C++ dasturlash tili C tiliga asoslangan. C esa o'z navbatida B va BCPL tillaridan kelib chiqqan. BCPL 1967 yilda Martin Richards tomonidan tuzilgan va operatsion sistemalarni yozish uchun mo'ljallangan edi. Ken Thompson
o'zining B tilida BCPL ning ko'p hossalarini kiritgan va B da UNIX operatsion sistemasining birinchi versiyalarini yozgan. BCPL ham, B ham tipsiz til bo'lgan. Yani o'garuvchilarning ma'lum bir tipi bo'lmagan - har bir o'zgaruvchi kompyuter hotirasida faqat bir bayt yer
egallagan. O'zgaruvchini qanday sifatda ishlatish esa, yani butun sonmi, kasrli sonmi yoki harfdekmi, dasturchi vazifasi bo'lgan.

C tilini Dennis Ritchie B dan keltirib chiqardi va uni 1972 yili ilk bor Bell Laboratoriyasida, DEC PDP-11 kompyuterida qo'lladi. C o'zidan oldingi B va BCPL tillarining juda ko'p muhim tomonlarini o'z ichiga olish bilan bir qatorda o'zgaruvchilarni tiplashtirdi va bir qator boshqa yangiliklarni kiritdi. Boshlanishda C asosan UNIX sistemalarida keng tarqaldi. Hozirda operatsion sistemalarning asosiy qismi C/C++ da yozilmoqda. C mashina arhitekturasiga bog'langan tildir. Lekin yahshi rejalashtirish orqali dasturlarni turli kompyuter platformalarida ishlaydigan qilsa bo'ladi.

1983 yilda, C tili keng tarqalganligi sababli, uni standartlash harakati boshlandi. Buning uchun Amerika Milliy Standartlar Komiteti (ANSI) qoshida X3J11 tehnik komitet tuzildi. Va 1989 yilda ushbu standart qabul qilindi. Standartni dunyo bo'yicha keng tarqatish maqsadida 1990 yilda ANSI va Dunyo Standartlar Tashkiloti (ISO) hamkorlikda C ning ANSI/ISO 9899:1990 standartini qabul qilishdi. Shu sababli C da yozilgan dasturlar kam miqdordagi o'zgarishlar yoki umuman o'zgarishlarsiz juda ko'p kompyuter platformalarida ishlaydi.

C++ 1980 yillar boshida Bjarne Stroustrup tomonidan C ga asoslangan tarzda tuzildi. C++ juda ko'p qo'shimchalarni o'z ichiga olgan, lekin eng asosiysi u ob'ektlar bilan dasturlashga imkon beradi.

Dasturlarni tez va sifatli yozish hozirgi kunda katta ahamiyat kasb etmoda. Buni ta'minlash uchun ob'ektli dasturlash g'oyasi ilgari surildi. Huddi 70-chi yillar boshida strukturali dasturlash kabi, programmalarni hayotdagi jismlarni modellashtiruvchi ob'ektlat orqali tuzish dasturlash sohasida inqilob qildi.

C++ dan tashqari boshqa ko'p ob'ektli dasturlshga yo'naltirilgan tillar paydo bo'ldi. Shulardan eng ko'zga tashlanadigani Xerox ning Palo Altoda joylashgan ilmiy-qidiruv markazida (PARC) tuzilgan Smalltalk dasturlash tilidir. Smalltalk da hamma narsa ob'ektlarga asoslangan. C++ esa gibrid tildir. Unda C ga o'hshab strukturali dasturlash yoki yangicha, ob'ektlar bilan dasturlash mumkin. Yangicha deyishimiz ham nisbiydir. Ob'ektli dasturlash falsafasi paydo bo'lganiga ham yigirma yildan oshayapti.

C++ funksiya va ob'ektlarning juda boy kutubhonasiga ega. Yani C++ da dasturlashni o'rganish ikki qismga bo'linadi. Birinchisi bu C++ ni o'zini o'rganish, ikkinchisi esa C++ ning standart kutubhonasidagi tayyor ob'ekt/funksiyalarni qo'llashni o'rganishdir.

C++ DA DASTURLASHNING ASOSIY QISMLARI

C++ sistemasi asosan quyidagi qismlardan iborat. Bular dasturni yozish redaktori, C++ tili va standart utubhonalardir. C++ dasturi ma'lum bir fazalardan o'tadi. Birinchisi dasturni yozish va tahrirlash, ikkinchisi
preprosessor amallarini bajarish, kompilyatsiya, kutubhonalardagi ob'ekt va funksiyalarni dastur bilan bog'lash (link), hotiraga yuklash (load) va bajarish (execute).

C++ DA BIRINCHI PROGRAMMA

```
//C++ dagi ilk dasturimiz
/*Ekranga yozuv chiqarish*/

# include <iostream.h>
 int main()
{
cout << "Hello World!\n";
return 0; //Dastur kutulganidek tugaganinig belgisi.
}
```

Ekranda:

```
Hello World!
```

Dasturni satrma-satr tahlil qilaylik. C++ da ikki tur sharhlar mavjud. /* bilan boshlanib, */ bilan tugaydigani bir necha satrni egallashi mumkin. Yani bu belgilar orasida qolgan hamma yozuv sharh hisoblanadi. Bu tur sharh C dan qolgan. C++ yangi ko'rinishdagi sharhlar ham kiritilgan. Bu // bilan boshlanadi va kuchi shu satr ohirigacha saqlanadi. Sharhlar yoki boshqacha qilib aytganda kommentariylar kompilyator tomonidan hisobga olinmaydi va hech qanday mashina ijro kodiga aylantirilmaydi. Sharhlar kerakli joyda, funksiyalardan oldin, o'zgaruvchilar e'lonidan keyin yozilganda, dasturni tushunish ancha osonlashadi va keyinchalik programma ishlash mantig'ini esga solib turadi.

include <iostream.h> bu preprosessorga beriladigan buyruqdir.

Preprosessor kompilyatsiyadan oldin fayllarni ko'rib chiqadi va kerakli amallarni bajaradi. Unga tegishli bo'lgan buyruqlar # belgisi bilan boshlanadi, lekin buyruq ohiriga nuqta-vergul (;) qoyilmaydi. Bu yerda **include** (kiritmoq, qamrab olmoq) buyrug'i **iostream.h** faylini asosiy dasturimiz ichiga kiritadi. Bu fayl ichida biz ishlatayotgan cout oqim (stream) ob'ektining e'loni berilgan. C++ tilida ekran yoki klaviyaturadan kirish/chiqishni bajarmoqchi bo'lgan barcha dasturlar ushbu boshliq (header) faylni yoki uning yangi ko'rinishini **include** bilan o'z ichiga olishi kerak. Bu kabi fayllarni biz bundan keyin e'lon fayllari deb ataymiz. Chunki bu fayllar ichida funksiya va ob'ektlarning o'zi, yani tanasi berilmay faqatgina e'loni beriladi.

include buyrug'i bir necha hil yo'l bilan qo'llanilishi mumkin:

1. include <stdio.h>
2. include <iostream.h>
3. include "meningfaylim.h"

Direktivalar- funksiylar kutubxonasini chaqirish.Ular maxsus **include** katologida joylashgan va .h fayllar deb nomlanadi. C++ tilida masalaning qoyilishiga qarab kerakli include lar chaqiriladi. Bu esa dasturning xotirada egalaydigan joyini minimallashtiradi.

Masalan: malumotlarni kiritish-chiqarish proseduralari uchun birinchi usulda e'lon fayli <> qavslari ichida yoziladi. Bunda C++ sistemasi ushbu faylni oldindan belgilangan kataloglar ichidan qidiradi. Bu usul bilan
asosan standart kutubhona fayllari qo'llaniladi. Ikkinchi usulda, fayl nomi qo'shtirnoqlarga olinganda, kiritilishi kerak bo'lgan fayl joriy katalogdan qidiriladi. Bu yo'l bilan dasturchi o'zi yozgan e'lon fayllarini kiritadi.

Shuni aytib o'tish kerakki, C++ ning 1998 yili qabul qilingan standartiga ko'ra, ushbu e'lon fayllari yangi ko'rinishga ega, ular .h bilan tugamaydi.
Bunda, misol uchun: bizning <iostream.h> faylimiz iostream, C dan kelgan math.h esa cmath nomiga ega. Biz bu o'zgarishlarga keyinroq qaytamiz, hozircha esa eski tipdagi e'lon fayllaridan foydalanib turamiz.

int main() har bir C++ dasturining qismidir. main dan keyingi ()qavslar C++ ning funksiya deb ataluvchi blokining boshlanganligini bildiradi. C++ dasturi bir yoki bir necha funksiyalardan iborat. Va shulardan aniq bitta funksiya **main** deb atalishi shart. Bunda main dastur ichida keladigan birinchi funksiya bo'lmasligi ham mumkin. Operatsion sistema dastur ijrosini **main()** funksiyasidan boshlaydi.

main() dan oldin kelgan **int** esa **main** funksiyasidan qaytish qiymati tipini belgilaydi. Bunda int **integer**, yani butun son deganidir. **main()** ning qaytargan qiymati operatsion sistemaga boradi.

{ qavs funksiya va boshqa bloklar tanasini boshlaydi. Blokni yopish uchun } qavsi ishlatilinadi.

cout << "Hello World!\n"; satri C++ da **ifoda** deb ataladi. C++ dagi har bir ifoda ; (nuqta-vergul) bilan tugatilishi shart. Ortiqcha ; bo'sh ifoda deyiladi. Uni qo'yish dastur tezligiga ta'sir qilmaydi.

Kirish va chiqish (Input/Output), yani dasturga kerakli ma'lumotlarni kiritish va ular ustida dastur tomonidan bajarilgan amallar natijalarini olish C++ da oqim ob'ektlari orqali bajarilishi mumkin. Lekin kirish/chiqishni C dagi kabi funksiyalar bilan ham amalga oshirsa bo'ladi.

C++ falsafasiga ko'ra har bir kirish/chiqish jihozi (ekran, printer, klaviatura...) baytlar oqimi bilan ishlagandek qabul qilinadi. Yuqoridagi ifoda bajarilganda bizning "Hello World!" gapimiz standart chiqish oqimi ob'ekti **cout** ga (cout - console out) jo'natiladi. Normal sharoitda bu oqim ekranga ulangandir.
C++ da satrlar (string) qo'shtirnoqlar ('') orasida bo'ladi. Bitta harfli literalar esa bitta tirnoq - apostrof (') ichiga olinadi. Misol uchun: 'A', '$'. Bitta harf yoki belgini qo'shtirnoq ichiga olsa u satr kabi qabul qilinadi.
<<operatori oqimga kiritish operatori deyiladi. Programma ijro etilganda <<operatorining o'ng tomonidagi argument ekranga yuboriladi. Bunda ekranga qo'shtirnoq ("...") ichidagi narsa bosib chiqariladi. Lekin e'tibor bersak, \n belgisi bosilmadi. \ (teskari kasr - backslash) belgisi mahsus ma'noga ega. U o'zidan keyin kelgan belgi oqim buyrug'i yoki manipulyatori ekanligini bildiradi. Shunda \ belgisi bilan undan keyin kelgan belgi buyruq ketma-ketligida aylanadi. Bularning ro'yhatini beraylik.
\n - Yangi satr. Kursor yangi qator boshidan joy oladi.
\t - Gorizontal tabulyatsiya (kursor bir-necha harf o'nga siljiydi).
\v - Vertikal tabulyatsiya (bir-necha satr tashlanib o'tiladi).
\r - Qaytish. Kursor ayni satr boshiga qaytadi, yani yangi satrga o'tmaydi.
\a - Kompyuter dinamiki chalinadi.
\\ - Ekranga teskari kasr belgisini bosish uchun qo'llaniladi.
\" - Ekranga qo'shtirnoq belgisini bosish uchun qo'llaniladi.
return 0; (return - qaytmoq) ifodasi **main()** funksiyasidan chiqishning asosiy yo'lidir. 0 (nol) qiymatining qaytarilishi operatsion sistemaga ushbu dastur normal bajarilib tugaganini bildiradi. **return** orqali qaytadigan qiymat tipi funksiya e'lonidagi qaytish tipi bilan bir hil bo'lishi kerak. Bizda bu e'lon **int main(){...}** edi. Va 0 **int** tipiga mansubdir. Bundan keyin **return** orqali qaytarilayotgan ifodani qavs ichiga olamiz. Misol uchun **return (6)**. Bu qavslar majburiy emas, lekin bizlar ularni programmani o'qishda qulaylik uchun kiritamiz.

BOSHQA BIR DASTUR

```
//Ushbu dastur ikki butun sonni ko'paytiradi.
# include <iostream.h>
int main()
{
int sonA, sonB; //o'zgaruvchi e'lonlari
int summa;      //e'lon
cout << "Birinchi sonni kiriting: ";
cin >> sonA; //Birinchi sonni o'qish...
cout << "Ikkinchi sonni kiriting: ";
cin >> sonB; //Ikkinchi sonni o'qish...
```

```
summa = sonA * sonB;
cout << summa << endl;
cout << "sonA * sonB = " << sonA * sonB << endl;
return (0);
}
Ekranda:
Birinchi sonni kiriting: 4
Ikkinchi sonni kiriting: 6
24
sonA * sonB = 24
```

int son A, son B; ifodasi int tipidagi, yani integer butun son) bo'lgan ikkita o'zgaruvchini e'lon declaration) qildik. Agar o'zgaruvchilar tipi bir hilda bo'lsa, yuqoridagi kabi ularni ketma-ket, vergul bilan ayirib yozsak bo'ladi. Keyingi satrda esa int summa; bilan summa nomli o'zgaruvchini e'lon qildik.
cout << "Birinchi sonni kiriting: "; ifodasi bilan ekranga nima qilish kerakligini yozib chiqdik.cin>>son A; amali cin kirish oqimi ob'ekti orqali son A o'zgaruvchisiga klaviaturadan qiymat kiritmoqda. Sonni yozib bo'lgandan so'ng Enter ni bosamiz. Normal sharoitda kirish oqimi klaviaturaga bog'langan. Shu tariqa sonB ga ham qiymat berdik. Keyin esa summa = sonA * sonB; bilan biz ikki o'zgaruvchini ko'paytirib, ko'paytma qiymatini summa ga beryapmiz. Bu yerdagi "=" va "*" operatorlar ikki argumentli operatorlar deyiladi, chunki ular ikkita operand yoki boshqacha qilib aytganda kirish qiymatlari bilan ishlaydi. Operatorlardan oldin va keyin bo'sh joy qoldirsak, o'qishni osonlashtirgan bo'lamiz.

Ekranga javobni chiqarganda, **cout** ga tayyor natijani (summa) yoki matematik ifodaning o'zini berishimiz mumkin. Ohirgi **cout** ga bir-necha argumentni berdik.
endl (end line - satrni tugatish) bu oqim manipulyatoridir (stream manipulator). Ba'zi bir sistemalar chiqish oqimiga yo'naltirilgan ma'lumotlarning ma'lum bir miqdori yig'ilguncha ushbu ma'lumotlarni ekranga bosib chiqarmay, buferda saqlashadi. Va o'sha chiqish buferi to'lgandan keyingina ma'lumotlarni ekranga yuborishadi. Buning sababi shuki, ekranga bosish nisbatan vaqt jihattan qimmat amaldir. Agar ma'lumotlar yig'ilib turib, bittada chiqarilsa, dastur ancha tez ishlaydi. Lekin biz yuqoridagi dasturdagi kabi qo'llanuvchi bilan savol-javob qiluvchi programmada yo'l-yo'riqlarimizni berilgan paytning o'zida ekranga bosib chiqarilishini hohlaymiz. Shu sababli biz **endl** ni ishlatishimiz kerak.
endl ni biz "\n" buyrug'iga tenglashtirishimiz mumkin. Yani **endl** ni ishlatganimizda, bufer yoki boshqacha qilib aytganda, hotiradagi ma'lumotni vaqtinchalik saqlanish joyidagi ma'lumot ekranga bosib chiqarilgandan so'ng, kursor yangi satr boshiga ko'chadi. Agar biz buferni bo'shatmoqchi-yu, lekin kursorni joyida saqlab qolmoqchi bo'lsak, **flash** manipulyatorini ishlatishimiz lozim.
Ifodamizga qaytaylik. **cout** << "sonA * sonB = " << sonA * sonB << endl; ifodasida chiqish ob'ekti bitta, lekin biz unga uchta narsani yubordik. Buni biz oqimga ma'lumotlarni chiqarishni kaskadlash, zanjirlash yoki konkatenatsiya qilish deb ataymiz. Ayni amalni **cin** (console in) kirish oqimi uchun ham bajara olamiz.

Hisob-kitoblar chiqish ifodasi ichida ham bajarilishi mumkin, cin << sonA *sonB << endl; bunga misol. Agar bu yo'lni tutganimizda, summa o'zgaruvchisi kerakmas bo'lib qolardi. Ushbu dasturda bizda yangi bo'lgan narsalardan biri bu o'zgaruvchi (variable) tushunchasidir. O'zgaruvchilar kompyuter hotirasidagi joylarga ko'rsatib turishadi. Har bir o'zgaruvchi ism, tip, hotirada egallagan joy kattaligi va qiymatga egadir. O'zgaruvchi ismi katta-kichik harf, son va past tiredan (_ - underscore) iboratdir. Lekin sondan boshlana olmaydi. C/C++ da katta-kichik harf, yani harflar registri farqlanadi. Misol uchun A1 va a1 farqli ismlardir.

C++ DA ARIFMETIK AMALLAR

Ko'p programmalar ijro davomida arifmetik amallarni bajaradi. C++ dagi amallar quyidagi jadvalda berilgan. Ular ikkita operand bilan ishlatildi.

C++ dagi amal	Arifmetik operator	Algebraik ifoda	C++ dagi ifodasi:
Qo'shish	+	h+19	h+19
Ayirish	-	f-u	f-u
Ko'paytirish	*	sl	s*l
Bo'lish	/	v/d, vöd	v/d
Modul olish	%	k mod 4	k%4

Bularning ba'zi birlarinig hususiyatlarini ko'rib chiqaylik. Butun sonli bo'lishda, yani bo'luvchi ham, bo'linuvchi ham butun son bo'lganda, javob butun son bo'ladi. Javob yahlitlanmaydi, kasr qismi tashlanib yuborilib, butun qismining o'zi qoladi.
Modul operatori (%) butun songa bo'lishdan kelib chiqadigan qoldiqni beradi. x%y ifodasi x ni y ga bo'lgandan keyin chiqadigan qoldiqni beradi. Demak, 7%4 bizga 3 javobini beradi. % operatori faqat butun sonlar bilan ishlaydi. Vergulli (real) sonlar bilan ishlash uchun **"math.h"** kutubhonasidagi **fmod** funksiyasini qo'llash kerak.

C++ da qavslarning ma'nisi huddi algebradagidekdir. Undan tashqari boshqa boshqa algebraik ifodalarning ketma-ketligi ham odatdagidek. Oldin ko'paytirish, bo'lish va modul olish operatorlari ijro ko'radi. Agar bir necha operator ketma-ket kelsa, ular chapdan o'nga qarab ishlanadi. Bu operatorlardan keyin esa qo'shish va ayirish ijro etiladi.
Misol keltiraylik. k = m * 5 + 7 % n / (9 + x);
Birinchi bo'lib m * 5 hisoblanadi. Keyin 7 % n topiladi va qoldiq (9 + x) ga bo'linadi. Chiqqan javob esa m * 5 ning javobiga qo'shiladi. Qisqasini aytsak, amallar matematikadagi kabi. Lekin biz o'qishni osonlashtirish uchun va hato qilish ehtimolini kamaytirish maqsadida qavslarni kengroq ishlatishimiz mumkin. Yuqoridagi misolimiz quyidagi ko'rinishga ega bo'ladi.

```
k = ( m * 5 ) + ( ( 7 % n ) / ( 9 + x ) );
```

MANTIQIY SOLISHTIRISH OPERATORLARI

C++ bir necha solishtirish operatorlariga ega.

Algebraik ifoda	C++ dagi operator	C++ dagi ifoda	Algebraik ma'nosi	tenglik guruhi
=	==	x==y	x tengdir y ga	
teng emas	!=	x!=y	x teng emas y ga	
solishtirish guruhi				
>	>	x>y	x katta y dan	
<	<	x<y	x kichkina y dan	
katta-teng	>=	x>=y	x katta yoki teng y ga	
kichik-teng	<=	x<=y	x kichik yoki teng y ga	

==, !=, >= va <= operatorlarni yozganda oraga bo'sh joy qo'yib ketish sintaksis hatodir. Yani kompilyator dasturdagi hatoni ko'rsatib beradi va uni tuzatilishini talab qiladi. Ushbu ikki belgili operatorlarning belgilarining joyini almashtirish, masalan <= ni =< qilib yozish ko'p hollarda sintaksis hatolarga olib keladi. Gohida esa != ni =! deb yozganda sintaksis hato vujudga ham, bu mantiqiy hato bo'ladi. Mantiqiy hatolarni kompilyator topa olmaydi. Lekin ular programma ishlash mantig'ini o'zgartirib yuboradi. Bu kabi hatolarni topish esa ancha mashaqqatli ishdir (! operatori mantiqiy inkordir). Yana boshqa hatolardan biri tenglik operatori (==) va tenglashtirish, qiymat berish operatorlarini (=) bir-biri bilan almashtirib qo'yishdir. Bu ham juda ayanchli oqibatlarga olib keladi, chunki ushbu hato aksariyat hollarda mantiq hatolariga olib keladi.

Yuqoridagi solishtirish operatorlarini ishlatadigan bir dasturni ko'raylik.

```
//Mantiqiy solishtirish operatorlari
  # include <iostream.h>
int main()
{
int s1, s2;
cout << "Ikki son kiriting: " << endl;
cin >> s1 >> s2;                    //Ikki son olindi.
if (s1 == s2) cout << s1 << " teng " << s2 << " ga" << endl;
if (s1 < s2) cout << s1 << " kichik " << s2 << " dan" << endl;
if (s1 >= s2) cout << s1 << " katta yoki teng " << s2 << " ga" << endl;
if (s1 != s2) cout << s1 << " teng emas " << s2 << " ga" << endl;
return (0);
}
```

Ekranda:

```
Ikki sonni kiriting: 74 33
74 katta yoki teng 33 ga
74 teng emas 33 ga
```

Bu yerda bizga yangi bu C++ ning if (agar) struktura-sidir. if ifodasi ma'lum bir shartning to'g'ri (true) yoki noto'g'ri (false)bo'lishiga qarab, dasturning u yoki bu blokini bajarishga imkon beradi. Agar shart to'g'ri bo'lsa, if dan so'ng keluvchi amal bajariladi. Agar shart bajarilmasa, u holda if tanasidagi ifoda bajarilmay, if dan so'ng keluvchi ifodalar ijrosi davom ettiriladi. Bu strukturaning ko'rinishi quyidagichadir:

```
if (shart) ifoda;
```

Shart qismi qavs ichida bo'lishi majburiydir.Eng ohirida keluvchi nuqta-vergul (;) shart qismidan keyin qo'yilsa (if (shart); ifoda;) mantiq hatosi vujudga keladi. Chunki bunda if tanasi bo'sh qoladi. Ifoda qismi esa shartning to'g'ri-noto'g'ri bo'lishiga qaramay ijro qilaveradi.

C++ da bitta ifodani qo'yish mumkin bo'lgan joyga ifodalar guruhini ham qo'yish mumkin. Bu guruhni {} qavslar ichida yozish kerak. if da bu bunday bo'ladi:

```
if (shart) {
  ifoda1;
  ifoda2;
 ...
  ifodaN;
}
```

Agar shart to'g'ri javobni bersa, ifodalar guruhi bajariladi, aksi taqdirda blokni yopuvchi qavslardan keyingi ifodalardan dastur ijrosi davom ettiriladi.

YANGI STILDAGI E'LON FAYLLARI VA ISMLAR SOHASI TUSHUNCHASI

C++ ning standarti .h bilan tugaydigan (stdio.h ...) standart kutubhona e'lon fayllarini yangittan nomlab chiqdi. Bunda .h qo'shimchasi olib tashlandi.

C dan qolgan fayllar ismiga esa c harfi qo'shildi.

Misol uchun:

```
iostream.h -> iostream
string.h   -> cstring
stdlib.h   -> cstdlib
time.h     -> ctime
```

C dan meros qolgan kutubhona 18 ta e'lon fayli orqali berilgan. C++ ga tegishli standart kutubhonada esa 32 ta e'lon fayl bor. Fayllarni yangittan belgilashdan maqsad kutubhodadagi funksiya va ob'ektlarni **std** deb ataluvchi ismlar sohasiga (namespace) kiritishdir.

Ismlar sohasining o'zi ham nisbatan yangi tushuncha. Ismlar sohasini alohida dastur qismlari deb faraz qilsak boladi. Boshqa-boshqa sohalarda ayni ismli funksiya, o'zgaruvchi nomlari va ob'ektlar berilishi mumkin. Va bunda hech qanday ismlar to'qnashuvi sodir bo'lmaydi. Misol uchun bizda global, **std** va fun::obj degan ism sohalari bo'lsin. Ularning har birining ichida esa **cout**
nomli ob'ekt aniqlangan bo'lsin. C++ da to'liq aniqlangan ism (fully qualified name) degan tushuncha bor. Shunga ko'ra har bir cout ob'ektinig to'liq ismi quyidagicha bo'ladi:

Ismlar sohasi	ob'ekt
global	::cout
std	std::cout
fun::obj	fun::obj::cout

:: operatori sohalarni bog'lash uchun qo'llaniladi. fun::obj nomli ismlar sohasida obj fun ichida joylashgan ism sohasidir.
Global ismlar sohasida aniqlangan funksiya va boshqa turdagi dastur birliklariga programmaning
istalgan yeridan yetishsa bo'ladi. Masalan global ismlar sohasida e'lon qilingan int tipidagi k ismli o'zgaruvchimiz bol'sa, uning ustidan dasturning hohlagan blokida amal bajarsak bo'ladi.

Ismlar sohasi mehanizmi dasturchilarga yangi kutubhonalarni yozish ishini ancha osonlashtiradi. Chunki yangi kutubhonada ayni ismlar qo'llanishiga qaramay, ismlar konflikti yuz bermaydi. Dastur yoki kutubhona yozganda yangi ismlar sohasini belgilash uchun namespace istalgan_ism {

```
...
foo();
int k;
String str;
...
}
```

deb yozamiz. Dasturimizda ushbu ismlar sohasida aniqlangan o'zgaruvchilarni ishlatish uchun ularning to'liq ismini yozishimiz kerak.
Masalan:

```
istalgan_ism::foo();
```

Ammo bu usul ancha mashaqqatli bo'ladi. Har bir funksiya yoki o'zgaruvchi oldiga uning to'liq aniqlangan ismini yozish ko'p vaqt oladi. Buning o'rniga biz

```
using namespace istalgan_ism;
```

deb yozib o'tsak kifoya. using (ishlatish, qo'llash) buyrug'i bizning ismlar sohamizni dasturimiz ichiga tanishtiradi. Eng asosiysi biz bu amalni sohada aniqlangan va biz qo'llamoqchi bo'lgan ismlarning ilk chaqirig'idan oldin yozishimiz kerak. C++ ning standart kutubhonasida aniqlangan ifodalarni qo'llash uchun biz
using namespace std; deymiz. Va albatta qo'llanilayotgan e'lon fayllari yangi tipda bo'lishi kerak. Endi bu tushunchalarni ishlatadigan bir dasturni keltiraylik.

```
//Yangi tipdagi e'lon fayllari va ismlar sohasini qo'llash.
# include <iostream>
using namespace std;
int main()
{
std::cout << "Hello!\n";
cout << "Qale!";
return (0);
}
```

Ekranda:

```
Hello!
Qale!
```

std::cout << "Hello\n"; satrida biz chiqish oqimi ob'ekti cout ning to'liq ismini qo'lladik. Keyingi satrda esa biz yana ayni ob'ektni ishlatdik, lekin endi uni to'liq atab o'tirmadik, chunki biz std ismlar sohasini using bilan e'lon qilib bo'ldik. Ismlarni global ismlar sohasida e'lon qilish uchun ularni blok va funkisiyalar tashqarisida aniqlash kerak. Masalan:

```
# include <stdio.h>
int i;
int main()
{
...
int k;
...
return (0);
}
```

Bu yerda i global ismlar sohasida joylashgan, k esa main() funksiyasiga tegishli. O'zgaruvchilarni global ism sohasida aniqlashning boshqa yo'li, ularni nomsiz ismlar sohasida belgilashdir. Yani:

```
namespace {
int j;
}
```

j o'zgaruvchisi global boldi. Uni ishlatish uchun:

```
::j = ::j + 7;
```

:: operatorini qo'llashimiz mumkin, yoki oddiygina qalib faqat o'zini:

j = j * 9;

kabi yozishimiz mumkin. Ammo agar biz ishlayotgan dastur blokida ayni ismli o'zgaruvchi bo'lsa, masalan j, unda ushbu lokal aniqlangan j bizning global j imizni berkitib qo'yadi. Biz j ni o'zini qo'llasak, lokal j ga murojat
qilgan bo'lamiz. Global j ni ishlatish uchun endi :: operatorini qo'llashga majburmiz. Bu mulohazalar umuman olganda boshqa ismlar sohalarini ishlatganimizda ham o'rinlidir.

2. BOSHQARUV IFODALARI

Bu bo'limda biz strukturali dasturlashning asosiy prinsip va qismlarini ko'rib chiqamiz. Ma'lum bir dasturni yozish uchun belgilangan qadamlarni bosib o'tish kerak. Masala aniqlangandan so'ng uni yechish uchun mo'ljallangan algoritm tuziladi. Keyin esa psevdokod yoziladi. Psevdokod algoritmda bajariladigan qadamlarni ko'rsatadi. Bunda faqat bajariladigan ifodalar ko'rib chiqiladi. Psevdokodda o'zgaruvchi e'lonlari yoki boshqa ma'lum bir dasturlash tiliga mansub bo'lgan yordamchi amallar bo'lmaydi. Psevdo kodni yozish dasturlashni ancha osonlashtiradi, algoritm mantig'ini tushunishga va uni rivojlanritishga katta yordam beradi. Misol uchun bir dasturning rejasi va psevdo kodi 3-4 oy yozilgan bo'lsa va yuqori darajada detallashtirilgan bo'lsa, ushbu dasturning C++ yoki boshqa tildagi kodini yozish 2-3 hafta vaqt oladi halos. Bu yozilgan programmada hato ancha kam bo'ladi, uni keyinchalik takomillashtirish arzonga tushadi. Hozirgi paytda dastur o'zgarishi favqulotda hodisa emas, balki zamon talabidir.

DASTUR IJRO STRUKTURALARI

Asosan dasturdagi ifodalar ketma-ket, navbatiga ko'ra ijro etiladi. Gohida bir shart bajarilishiga ko'ra, ijro boshqa bir ifodaga o'tadi. Navbatdagi emas, dasturning boshqa yerida joylashgan ifoda bajariladi. Yani sakrash yoki ijro ko'chishi vujudga keladi. 60-chi yillarga kelib, dasturlardagi ko'pchilik hatolar aynan shu ijro ko'chishlarining rejasiz ishlatilishidan kelib chiqishi ma'lum bo'ldi. Bunda eng katta aybdor deb bu ko'shishlarni amalga oshiruvchi goto(..ga bor)ifodasi belgilandi. **goto** dastur ijrosini deyarli istalgan yerga ko'chirib yuborishi mumkin. Bu esa programmani o'qishni va uning strukturasini murakkablashtirib yuboradi. Shu sababli "strukturali dasturlash" atamasi "goto ni yo'q qilish" bilan tenglashtirilardi. Shuni aytib o'tish kerakki,goto kabi shartsiz sakrash amallarini bajaruvchi ifodalar boshqa dasturlash tillarida ham bor. Tadqiqotlar shuni ko'rsatdiki, istalgan programma goto siz yozilishi mumkin ekan. goto siz yozish uslubi strukturali dasturlash deb nom oldi. Va bunday dastur yozish metodi katta iqtisodiy samara beradi. Strukturali dasturlash asosi shundan iboratki, har bir programma faqatgina uch hil boshqaruv strukturalaridan iboratdir.Bular ifodalarni ketma-ket ijro etish strukturasi (sequence structure), tanlash strukturasi (selection structure) va amalni qayta ijro etish strukturasidir (repetition structure).

Ifodalarni ketma-ket ijro etish strukturasi C++ tomonidan ta'minlanadi. Normal sharoitda C++ ifodalari dasturdagi navbatiga ko'ra bajariladi. Tanlash buyruqlari uchtadir. Bular **if, if/else** va **switch** dir. Qayta ijro etish buyruqlari gurugiga ham uchta a'zo bor, bular **while, do/while** va **for**. Bularni har birini keyinroq tahlil qilib chiqamiz. Yuqoridagi buyruqlar nomlari C++ dasturlash tilining mahsus so'zlaridir. Dasturchi bu so'zlarni o'zgaruvchi yoki funksiyalar nomi sifatida qo'llashi ta'qiqlanadi. Quyida C++ ning ajratilgan so'zlarining to'liq ro'yhati berilgan.

C++ va C ga tegishli:

```
auto      do      goto      signed     unsigned
break     double  if        sizeof     void
case      else    int       static     volatile
char      enum    long      struct     while
const     extern  register  switch
continue  float   return    typedef
default   for     short     union
```

Faqat C++ ga qarashli:

```
asm           explicit   operator        this       virtual
bool          false      private         throw      wchar_t
catch         friend     protected       true
class         inline     public          try
const_cast    mutable    reinterpret_cast  typeid
delete        namespace  static_cast     typename
dynamic_cast  new        template        using
```

C++ dagi yetita boshqaruv strukturasini aytib o'tdik. Ular bittagina boshlanish nuqtasiga va bittagina chiqish nuqtasiga egadirlar. Demak biz bu dastur bo'laklarini ketma-ket ulab ketishimiz mumkin. Boshqaruv trukturalari-
ning bu kabi ulanishini devorning g'ishtlarini ustma-ust
qalashga ham taqqoslasak bo'ladi. Yoki biz bu bloklarni bir-birining ichiga joylashtirishimiz mumkin. Bu kabi qo'llashish ikkinchi uslub bo'ladi. Mana shu ikki yo'l bilan bog'langan yetita blok yordamida biz istalgan dasturimizni yoza olamiz.

if STRUKTURASI

Biz shartga ko'ra bir necha harakat yo'lidan bittasini tanlaymiz. Misol uchun: agar bolaning yoshi 7 ga teng yoki katta bo'lsa u maktabga borishi mumkin bo'lsin. Buni C++ da if ni qo'llab yozaylik.

```
if (yosh >= 7)
  maktab();
```

Bu yerda shart bajarilishi yoki bajarilmasligi mumkin. Agar yosh o'zgaruvchisi 7 ga teng yoki undan katta bo'lsa shart bajariladi va maktab() funksiyasi chaqiriladi. Bu holat true (to'g'ri) deyiladi. Agar yosh 7 dan kichik bo'lsa, maktab() tashlab o'tiladi. Yani false (noto'g'ri) holat yuzaga keladi. Biz shart qismini mantiqiy operator-larga asoslanganligini ko'rib chiqqan edik. Aslida esa shartdagi ifodaning ko'rinishi muhim emas – agar ifodani nolga keltirish mumkin bo'lsa false bo'ladi, noldan farqli javob bo'lsa, musbatmi, manfiymi, true holat paydo bo'ladi va shart bajariladi. Bunga qo'shimcha qilib o'tish kerakki,

C++ da mahsus bool tipi mavjud. Bu tipdagi o'zgaruvchilarning yordamida bul (mantiqiy) arifmetikasini amalgam oshirish mumkin. bool o'zgaruvchilar faqat true yoki false qiymatlarini olishlari mumkin.

if/else STRUKTURASI

if ni qo'llaganimizda ifoda faqat shart haqiqat bo'lgandagina bajariladi, aks holda tashlanib o'tiladi. if/else yordamida esa shart bajarilmaganda (false natija chiqqanda) else orqali boshqa bir yo'ldan borishni belgilash mumkin. Misolimizni takomillashtirsak. Bola 7 yosh yoki undan katta bo'lsa maktabga, 7 dan kichkina bo'lsa bog'chaga borsin.

```
if (yosh >= 7)
  maktab(); //nuqta-vergul majburiydir
else
  bogcha();
```

Yuqorida if ga tegishli bo'lgan blok bitta ifodadan (maktab()) iborat. Shu sababli nuqta-vergul qo'yilishi shart. Buni aytib o'tishimizning sababi, masal Pascalda hech narsa qo'yilmasligi shart. C++ da bitta ifosa turgan joyga ifodalar guruhini {} qavslarda olingan holda qo'ysa bo'ladi. Masalan:

```
if (yosh >= 7){
  cout << "Maktabga!\n";
  maktab();
 }
else{
  cout << "Bog'chaga!\n";
  bogcha();
 }
```

Aslida har doim {} qavslarni qo'yish yahshi odat hisoblanadi; keyinchalik bir ifoda turgan joyga qo'shimcha qilinganda qavslardan biri unutilib qolmaydi.

Strukrurali dasturlashning yana bir harakterli joyi shundaki tabulyatsiya, bo'sh joy va yangi satrlar ko'p qo'llaniladi. Bu programmani o'qishni osonlashtirish uchun qilinadi. C++ uchun bo'sh joyning hech ahamiyati yo'q, lekin dasturni tahrir qilayatgan odamga buyruqlar guruhini, bloklarni tabulyatsiya yordamida ajratib bersak, unga katta yordam bo'ladi. Yuqoridagini quyidagicha ham yozish mumkin:

```
if(yosh>=7){cout<<"Maktabga!\n";maktab()}else{cout<<"Bog'chaga!\n";bogcha()};
```

Biroq buni o'qish ancha murakkab ishdir.

C++ da if/else strukturasiga o'hshash ?: shart operatori (conditional operator) ham bordir. Bu C++ ning bittagina uchta argument oluvchi operatori. Uch operand va shart operatori shart ifodasini beradi. Birinchi operand orqali shartimizni beramiz. Ikkinchi argument shart true (haqiqat) bo'lib chiqqandagi butun shart ifodasining javob qiymatidir. Uchinchi operand shartimiz bajarilmay (false) qolgandagi butun shart ifodasining qiymatidir. Masalan:

```
bool bayroq;
int yosh = 10;
bayroq = ( yosh >= 7 ? true : false );
```

Agar yosh 7 ga teng yoki katta bo'lsa, bool tipidagi o'zgaruvchimiz true qiymatini oladi, aks taqdirda false bo'ladi. Shart operatori qavslar ichida bo'lishi zarur, chunki uning kuchi katta emas. Javob qiymatlar bajariladigan funksiyalar ham bo'lishi mumkin:

```
yosh >= 7 ? maktab() : bogcha();
```

if/else strukturalarini bir-birining ichida yozishimiz mumkin. Bunda ular bir-biriga ulanib ketadi. Misol uchun tezlikning kattaligiga qarab jarimani belgilab beruvchi blokni yozaylik.

```
if (tezlik > 120)
  cout << "Jarima 100 so'm";
else if (tezlik > 100)
  cout << "Jarima  70 so'm";
else if (tezlik > 85)
  cout << "Jarima  30 so'm";
else
  cout << "Tezlik  normada";
```

Agar tezlik 120 dan katta bo'lsa birinchi if/else strukturasining haqiqat sharti bajariladi. Va bu holda albatta tezlik o'zgaruvchimizning qiymati ikkinchi va uchinchi if/else imizni ham qoniqtiradi. Lekin solishtirish ulargacha bormaydi, chunki ular birinchi if/else ning else qismida, yani noto'g'ri javob qismida joylashgandir. Solishtirish birinchi if/else da tugashi (aynan shu misolda)tanlash amalini tezlashtiradi. Yani bir-biriga bog'liq if/else lar alohida if struktura-lari blokidan tezroq bajarilishi mumkin, chunki birinchi holda if/else blokidan vaqtliroq chiqish imkoni bor.Shu sababli ich-ichiga kirgan if/else lar guruhida true bo'lish imkoni ko'proq bo'lgan shartlarni oldinroq tekshirish kerak.

switch STRUKTURASI

if-else-if yordami bilan bir necha shartni test qilishimiz mumkin. Lekin bunday yozuv nisbatan o'qishga qiyin va ko'rinishi qo'pol bo'ladi. Agar shart ifoda butun son tipida bo'lsa yoki bu tipga keltirilishi mumkin bo'lsa, biz **switch** (tanlash) ifodalarini ishlata olamiz.

switch strukturasi bir necha case etiketlaridan (label) va majburiy bo'lmagan default etiketidan iboratdir. Etiket bu bir nomdir. U dasturnig bir nuqtasidaga qo'yiladi. Programmaning boshqa yeridan ushbu etiketga o'tishni bajarish mumkin. O'tish yoki sakrash goto bilan amalga oshiriladi, switch blokida ham qo'llaniladi.

5 lik sistemadagi bahoni so'zlik bahoga o'tqizadigan blokni yozaylik.

```
int baho;
baho = 4;
```

```
switch (baho) {
   case 5: cout << "A'lo";
     break;
  case 4: cout << "Yahshi";
     break;
  case 3: cout << "Qoniqarli";
     break;
  case 2:
  case 1: cout << "A'lo";
     break;
  default: cout << "Baho hato kiritildi!";
     break;
}
```

switch ga kirgan o'zgaruvchi (yuqorigi misolda baho) har bir case etiketlarining qiymatlari bilan solishtirilib chiqiladi. Solishtirish yuqoridan pastga bajariladi. Shartdagi qiymat etiketdagi qiymat bilan teng bo'lib chiqqanda ushbu case ga tegishli ifoda yoki ifodalar bloki bajariladi. So'ng break (buzmoq, tugatmoq) sakrash buyrug'i bilan switch ning tanasidan chiqiladi. Agar break qo'yilmasa, keyingi etiketlar qiymatlari bilan solishtirish bajarilmasdan ularga tegishli ifodalar ijro ko'raveradi. Bu albatta biz istamaydigan narsa. default etiketi majburiy emas. Lekin shart chegaradan tashqarida bo'lgan qiymatda ega bo'lgan hollarni diagnostika qilish uchun kerak bo'ladi.

case va etiket orasida bo'sh joy qoldirish shartdir. Chunki, masalan, case 4: ni case4: deb yozish oddiy etiketni vujudga keltiradi, bunda sharti test qilinayotgan ifoda 4 bilan solishtirilmay o'tiladi.

while TAKRORLASH STRUKTURASI

Takrorlash strukturasi bir ifoda yoki blokni ma'lum bir shart to'g'ri (true) bo'lishi davomida qaytarish imkonini beradi. Qaytarilayatgan ifoda shartga ta'sir ko'rsati-shishi kerak. Ma'lum bir vaqt o'tkandan keyin shart false ga o'zgartilishi kerak. Bo'lmasam while (davomida) tugatilmaydi. while faqat o'zidan keyin kelgan ifodaga ta'sir qiladi. Agar biz bir guruh amallarni qaytarmoqchi bo'lsak, ushbu blokni {} qavslar ichiga olishimiz kerak. Shart takrorlanuvchi blokning boshida tekshirilgani sababli, agar shart noto'g'ri bo'lib chiqsa, blokni hech ijro ko'rmasligi ham mumkin.

10 ning faktorialini hisoblovchi dastur blokini keltiraylik.

```
int factorial = 1;
int son = 1;
while (son < 11) {
  factorial = factorial * son;
  son = son + 1;
}
```

Bu yerda javobimiz factorial o'zgaruvchimizda saqlanmoqda. son o'zgaruvchimiz har takrorlanishda birga orttirilmoqda. son 11 ga yetganida while dagi shart false bo'ladi va takrorlanish tugatiladi. Yani son ning 11 qiymati javobga ta'sir ko'rsatmaydi. Biz qo'llagan son o'zgaruvchimiz sanovchi (counter)vazifasini bajaradi. Bu kabi o'zgaruvchilar vazifasiga ko'ra 1 yoki 0 ga tenglashtiriladi. Buni biz initsializatsiya deymiz. Initsializatsiya qilinmagan o'zgaruvchilar qiymatlari hotiradagi oldinroq ishlagan programmalar qoldiqlariga teng bo'ladi. Bu esa hatoga olib keladi. Shu sababli sanovchilarga boshlangish qiymat berib o'tilishi kerak.

do/while TAKRORLASH STRUKTURASI

do/while ifodasi while strukturasiga o'hshashdir. Bitta farqi shundaki while da shart boshiga tekshiriladi. do/while da esa takrorlanish tanasi eng kamida bir marta ijro ko'radi va shart strukturaning so'ngida test qilinadi. Shart true bo'lsa blok yana takrorlanadi. Shart false bo'lsa do/while ifodasidan chiqiladi. Agar do/while ichida qaytarilishi kerak bo'lgan ifoda bir dona bo'lsa {} qavslarning keragi yo'qdir. Quyidagicha bo'ladi:

```
do
  ifoda;
while (shart);
```

Lekin {} qavslarning yo'qligi dasturchini adashtirishi mumkin. Chunki qavssiz do/while oddiy while ning boshlanishiga o'hshaydi. Buni oldini olish uchun {} qavslarni har doim qo'yishni tavsiya etamiz.

```
int k = 1;
do {
  k = k * 5;
} while ( !(k>1000) );
```

Bu blokda 1000 dan kichik yoki teng bo'lgan eng katta 5 ga karrali son topilmoqda. while shartini ozroq o'zgarti-rib berdik, ! (not - inkor) operatorining ishlashini misolda ko'rsatish uchun. Agar oddiy qilib yozadigan bo'lsak, while shartining ko'rinishi bunday bo'lardi: while (k<=1000); Cheksiz takrorlanishni oldini olish uchun shart ifodasining ko'rinishiga katta e'tibor berish kerak. Bir nuqtaga kelib shart true dan false qiymatiga o'tishi shart.

QIYMAT BERISH OPERATORLARI

Bu qismda keyingi bo'limlarda kerak bo'ladigan tushuncha-larni berib o'tamiz.C++ da hisoblashni va undan keyin javobni o'zgaruvchiga beruvchi bir necha operator mavjuddir. Misol uchun:

k = k * 4; ni
k *= 4;
deb yozsak bo'aladi.
Bunda *= operatorining chap argumenti o'ng argumentga qo'shiladi va javob chap argumentda saqlanadi. Biz har bir operatorni ushbu qisqartirilgan ko'rinishda yoza olamiz (+=, -=, /=, *= %=). Ikkala qism birga yoziladi. Qisqartirilgan operatorlar tezroq yoziladi, tezroq kompilyatsiya qilinadi va ba'zi bir hollarda tezroq ishlaydigan mashina kodi tuziladi.

1 ga OSHIRISH VA KAMAYTIRISH OPERATORLARI (INCREMENT and DECREMENT) C++ da bir argument oluvchi inkrenet (++) va dekrement (--) operatorlari mavjuddir. Bular ikki ko'rinishda ishlatilinadi, biri o'zgaruvchidan oldin (++f - preinkrement, --d - predekrement), boshqasi o'zgaruvchidan keyin (s++ - postinkrement, s-- - postdekrement) ishlatilgan holi. Bularning bir-biridan farqini aytin o'taylik. Postinkrementda o'zgaruvchining qiymati ushbu o'zgaruvchi qatnashgan ifodada shlatilinadi va undan keyin qiymati birga oshiriladi. Preinkrementda esa o'zgaruvchining qiymati birga oshiriladi, va bu yangi qiymat ifodada qo'llaniladi. Predekrement va postdekrement ham aynan shunday ishlaydi lekin qiymat birga kamaytiriladi. Bu operatorlar faqatgina o'zgaruvchining qiymatini birga oshirish/kamaytirish uchun ham ishlatilinishi mumkin, yani boshqa ifoda ichida qo'llanilmasdan. Bu holda pre va post formalarining farqi yo'q.
Masalan:
++r;
r++;
Yuqoridagilarning funksional jihattan hech qanday farqi yo'q, chunki bu ikki operator faqat r ning qiymatini oshirish uchun qo'llanilmoqda. Bu operatorlarni oddiy holda yozsak:
r = r + 1;
d = d - 1;

Lekin bizning inkrement/dekrement operatorlarimiz oddiygina qilib o'zgaruvchiga bir qo'shish/ayirishdan ko'ra tezroq ishlaydi. Yuqoridagi operatorlarni qo'llagan holda bir dastur yozaylik.

```
//Postinkremet, preinkrement va qisqartirilgan teglashtirish operatrlari
# include <iostream.h>
int main()
{
int k = 5, l = 3, m = 8;
cout << k++ << endl;    //ekranga 5 yozildi, k = 6 bo'ldi.
l += 4;                 // l = 7 bo'ldi.
cout << --m << endl;    // m = 7 bo'ldi va ekranga 7 chiqdi.
m = k + (++l);          // m = 6 + 8 = 14;
return (0);
}
```

Dasturdagi o'zgaruvchilar e'lon qilindi va boshqangich qiymatlarni olishdi.
cout << k++ << endl; ifodasida ekranga oldin k ning boshlangich qiymati chiqarildi, keyin esa uning qiymati 1 da oshirildi. l += 4; da l ning qiymatiga 4 soni qo'shildi va yangi qiymat l da saqlandi. cout << --m << endl; ifodasida m ning qiymati oldin predekrement qilindi,va undan so'ng ekranga chiqarildi. m = k + (++l); da oldin l ning qiymati birga ishirildi va l ning yangi qiymati k ga qo'shildi. m esa bu yangi qiymatni oldi. Oshirish va kamaytirish operatorlari va ularning argumentlari orasida bo'shliq qoldirilmasligi kerak. Bu operatorlar sodda ko'rinishdagi o'zgaruvchi-larga nisbatan qo'llanilishi mumkin halos.
Masalan:
++(f * 5);
ko'rinish noto'g'ridir.

MANTIQIY OPERATORLAR

Bosqaruv strukturalarida shart qismi bor dedik. Shu paytgacha ishlatgan shartlarimiz ancha sodda edi. Agar bir necha shartni tekshirmoqchi bo'lganimizda ayri-ayri shart qismlarini yozardik. Lekin C++ da bir necha sodda shartni birlashtirib, bitta murakkab shart ifodasini tuzishga yordam beradigan mantiqiy operatorlar mavjuddir. Bilar mantiqiy VA - && (AND), mantiqiy YOKI - || (OR) va mantiqiy INKOR - ! (NOT). Bular bilan misol keltiraylik. Faraz qilaylik, bir amalni bajarishdan oldin, ikkala shartimiz (ikkitadan ko'p ham bo'lishi mumkin) true (haqiqat) bo'lsin.

```
if (i < 10 && l >= 20){...}
```

Bu yerda {} qavslardagi ifodalar bloki faqat i 10 dan kichkina va l 20 dan katta yoki teng bo'lgandagina ijro ko'radi.
AND ning (&&) jadvali:

ifoda1	ifoda2	ifoda1 && ifoda2
false (0)	false (0)	false (0)
true (1)	false (0)	false (0)
false (0)	true (1)	false (0)
true (1)	true (1)	true (1)

Bu yerda true ni yeriga 1, false ni qiymati o'rniga 0 ni qo'llashimiz mumkin.

Boshqa misol:

```
while (g<10 || f<4){
...
}
```

Bizda ikki o'zgaruvchi bor (g va f). Birnchisi 10 dan kichkina yoki ikkinchisi 4 dan kichkina bo'lganda while ning tanasi takrorlanaveradi. Yani shart bajarilishi uchun eng kamida bitta true bo'lishi kerak, AND da (&&) esa hamma oddiy shartklar true bo'lishi kerak.

OR ning (||) jadvali:

ifoda1	ifoda2	ifoda1 || ifoda2
false (0)	false (0)	false (0)
true (1)	false (0)	true (1)
false (0)	true (1)	true (1)
true (1)	true (1)	true (1)

&& va || operatorlari ikkita argument olishadi. Bulardan farqli o'laroq, ! (mantiqiy inkor) operatori bitta argumet oladi, va bu argumentidan oldin qo'yiladi. Inkor operatori ifodaning mantiqiy qiymatini teskarisiga o'zgartiradi. Yani false ni true deb beradi, true ni esa false deydi.

Misol uchun:

```
if ( !(counter == finish) )
  cout << student_bahosi << endl;
```

Agar counter o'zgaruvchimiz finish ga teng bo'lsa, true bo'ladi,bu true qiymat esa ! yordamida false ga aylanadi. false qiymatni olgan if esa ifodasini bajarmaydi. Demak ifoda bajarilishi uchun bizga counter finish ga teng bo'lmagan holati kerak. Bu yerda ! ga tegishli ifoda () qavslar ichida bo'lishi kerak. Chunki mantiqiy operator-lar tenglilik operatorlaridan kuchliroqdir. Ko'p hollarda ! operatori o'rniga mos keladigan mantiqiy tenglilik yoki solishtirish operatorlarini ishlatsa bo'ladi, masalan yuqoridagi misol quyidagi ko'rinishda bo'ladi:

```
if (counter != finish)
  cout << student_bahosi << endl;
```

NOT ning jadvali:

ifoda	!(ifoda)
false (0)	true (1)
true (1)	false (0)

for TAKRORLASH STRUKTURASI

for strukturasi sanovchi (counter) bilan bajariladigan takrorlashni bajaradi.Boshqa takrorlash bloklarida (while, do/while) takrorlash sonini control qilish uchun ham sanovchini qo'llasa bo'lardi, bu holda takrorlanish sonini o'ldindan bilsa bo'lardi, ham boshqa bir holatning vujudga kelish-kelmasligi orqali boshqarish mumkin edi. Ikkinchi holda ehtimol miqdori katta bo'ladi. Masalan qo'llanuvchi belgilangan sonni kiritmaguncha takrorlashni bajarish kerak bo'lsa biz while li ifodalar-ni ishlatamiz. for da esa sanovchi ifodaning qiymati oshirilib (kamaytirilib) borilvuradi, va chegaraviy qiymatni olganda takrorlanish tugatiladi. for ifodasidan keyingi bitta ifoda qaytariladi. Agar bir necha ifoda takrorlanishi kerak bo'lsa, ifodalar bloki {} qavs ichiga olinadi.

```
//Ekranda o'zgaruvching qiymatini yozuvchi dastur, for ni ishlatadi.
# include <iostream.h>
int main()
{
 for (int i = 0; i < 5; i++){
    cout << i << endl;
 }
 return (0);
}
```

Ekranda:

```
0
1
2
3
4
```

for strukturasi uch qismdan iboratdir. Ular nuqtavergul bilan bir-biridan ajratiladi. for ning ko'rinishi:

```
for( 1. qism ; 2. qism ; 3. qism ){
takror etiladigan blok
}
```

1. qism - e'lon va initsalizatsiya.
2. qism - shartni tekshirish (oz'garuvchini chegaraviy qiymat bilan solishtirish).
3.qism - o'zgaruvchining qiymatini o'zgartirish.

Qismlarning bajarilish ketma-ketligi quyidagichadir:

Boshida 1. qism bajariladi (faqat bir marta), keyin

2. qismdagi shart tekshiriladi va agar u true bo'lsa takrorlanish bloki ijro ko'radi, va eng ohirda 3. qismda o'zgaruvchilar o'zgartiriladi, keyin yana ikkinchi qismga o'tiladi. for strukturamizni while struktura bilan almashtirib ko'raylik:

```
for (int i = 0; i < 10 ; i++)
  cout << "Hello!"<< endl;
```

Ekranga 10 marta Hello! so'zi bosib chiqariladi. I o'zgaruvchisi 0 dan 9 gacha o'zgaradi. i 10 ga teng bo'lganda esa i < 10 sharti noto'g'ri (false) bo'lib chiqadi va for strukturasi nihoyasiga yetadi. Buni while bilan yozsak:

```
int i = 0;
while ( i<10 ){
```

```
  cout << "Hello!" << endl;
  i++;
}
```

Endi for ni tashkil etuvchi uchta qismninig har birini alohida ko'rib chiqsak.Birinchi qismda asosan takrorlashni boshqaradigan sanovchi (counter)o'zgaruvchi-
lar e'lon qilinadi va ularga boshlangich qiymatlar beriladi (initsalizatsiya). Yuqoridagi dastur misolida buni int i = 0; deb berganmiz. Ushbu qismda bir necha o'zgaruvchilarni e'lon qilishimiz mumkin, ular vergul bilan ajratilinadi. Ayni shu kabi uchinchi qismda ham bir nechta o'zgaruvchilarning qiyma-tini o'zgartirishimiz mumkin. Undan tashqari birinchi qismda for dan oldin e'lon qilingan o'zgaruvchilarni qo'llasak bo'ladi.

Masalan:

```
 int k = 10;
int l;
for (int m = 2, l = 0 ; k <= 30 ; k++, l++, ++m) {
cout << k + m + l;
}
```

Albatta bu ancha sun'iy misol, lekin u bizga for ifodasining naqadar moslashuvchanligi ko'rsatadi. for ning qismlari tushurib qoldirilishi mumkin.

Masalan:

```
 for(;;) {}
```

ifodasi cheksiz marta qaytariladi. Bu for dan chiqish uchun break operatorini beramiz. Yoki agar sanovchi sonni takrorlanish bloki ichida o'zgartirsak,for ning 3. qismi kerak emas. Misol:

```
 for(int g = 0; g < 10; ){
   cout << g;
   g++;
}
```

Yana qo'shimcha misollar beraylik.

```
 for (int y = 100; y >= 0; y-=5){
...
ifoda(lar);
...
}
```

Bu yerda 100 dan 0 gacha 5 lik qadam bilan tushiladi.

```
 for(int d = -30; d<=30; d++){
...
ifoda(lar);
...
}
```

60 marta qaytariladi.

for strukrurasi bilan dasturlarimizda yanada yaqinroq tanishamiz. Endi

1. qismda e'lon qilinadigan o'zgaruvchilarning hususiyati haqida bir og'iz aytib o'taylik. Standartga ko'ra bu qismda e'lon qilingan o'zgaruvchilarning qo'l-
lanilish sohasi faqat o'sha for strukturasi bilan chegaralanadi. Yani bitta blokda joylashgan for struk-turalari mavjud bo'lsa, ular ayni ismli o'zgaruvchilarni qo'llana ololmaydilar. Masalan quyidagi hatodir:

```
 for(int j = 0; j<20 ; j++){...}
...
for(int j = 1; j<10 ; j++){...} //hato!
```

j o'zgaruvchisi birinchi for da e'lon qilinib bo'lindi. Ikkinchi for da ishlatish mumkin emas. Bu masalani yechish uchun ikki hil yo'l tutish mumkin.

Birinchisi bitta blokda berilgan for larning har birida farqli o'zgaruvchilarni qo'llashdir. Ikkinchi yo'l for lar guruhidan oldin sanovchi vazifasini bajaruvchi bir o'zgaruvchini e'lon qilishdir. Va for larda bu o'zgaruv-
chiga faqat kerakli boshlangich qiymat beriladi halos.

```
 for ning ko'rinishlaridan biri, bo'sh tanali for dir.
 for(int i = 0 ; i < 1000 ; i++);
```

Buning yordamida biz dastur ishlashini sekinlashtirishimiz mumkin.

BOSHQARUV STRUKTURALARIDA continue VA break IFODALARINI QOLLASH

while, do/while, switch va **for** strukturalarida **break** operatorini qo'llaganimizda ushbu dastur bajarilishi ushbu strukturalaridan chiqib ketadi va navbatdagi kelayatgan ifodadan davom etadi. Bunda boshqaruv struk-turalaridagi breakdan keyin keluvchi ifodalar ijro ko'ra olmay qoladi.

Buni misolda ko'rsataylik.

```
//break va for ni qo'llash
# include <iostream.h>
int main()
```

```
{
 int h, k = 3;
 for(h = 0; h < 10 ; h++){
    cout << h << "   ";
    if (k == 6)
       break;
    cout << k++ << endl;
 }
 cout << "for dan tashqarida: " << h << "    " << k << endl;
 return (0);
}
```

Ekranda:

```
0   3
1   4
2   5
3
for dan tashqarida 3   6
```

if ning sharti bajarilgandan so'ng break dan keyin joylashgan cout << k++ << endl; ifodasi ijro ko'rmadi. Biz o'zgaruvchilarni for dan tashqarida ham qollamoqchi bo'lganimiz uchun, ularni for dan oldin e'lon qildik.

continue ifodasi while, do/while yoki for ichida qo'llanilganda, takrorlanish tanasida continue dan keyin kelayatgan ifodalar tashlanib o'tilib, takrorlanishning yangi sikli (iteratisyasi) boshlanadi. Buni programma qismi misolida ko'rib chiqaylik.

```
 ...
for (int e = 1 ; e<=10 ; ++e){
   if ( (e%2) == 0 ) //juft son bo'sa siklni o'tqizvor
      continue;
   cout << e << " ";
}
...
```

Ekranda:

```
 1 3 5 7 9
```

Bu yerda bir-ikkita aytib o'tiladigan nuqtalar bor. continue va **break** ni ishlatish strukturali dasturlash falsafasiga to'g'ri kelmaydi. Ular dasturni analiz qilishni murakkablashtirib yuboradi. Bular o'rniga strukturali dasturlash amallarini qo'llagan holda boshqaruv strukturalarining harakatini o'zgartirish mumkin. Lekin boshqa tarafdan albatta bu sakrash ifodalari ayni ishni bajaradigan strukturali dasturlash iboralaridan ko'ra ancha tezroq ishlaydi. Boshqaruv strukturalarini qo'llanilgan bir misol keltiraylik. Dastur futbol o'yinlarining nechtasida durang, nechtasida birinchi va nechtasida ikkinchi komanda yutganini sanaydi.

```
// while - switch - cin.get - EOF ga misol
# include <iostream.h>
 int main()
{
int natija   = 0,     // O'yin natijasi
    durang   = 0,     // duranglar soni
    birinchi = 0,     // birinchi komanda yutug'i
    ikkinchi = 0;     // ikkinchi komanda yutug'i
 cout << "Durang - d, birinchi komanda yutug'i - b,
       ikkinchi komanda yutug'i - i\n"
    << "Tugatish uchun - EOF." << endl;
 while ( ( natija = cin.get() ) != EOF ) {
 switch (natija) {
   case 'D':  // Katta harf uchun
   case 'd':  // kichkina harf uchun
      durang++;
      break;  //
    case 'B':
   case 'b':
      birinchi++;
      break;
    case 'I':
   case 'i':
      ikkinchi++;
      break;
    case '\n': //yangi satr
   case '\t': //tabulaytsiya
   case ' ' : //va bo'shliqlarga etibor bermaslik
      break;
   default: // qolgan hamma harflarga javob:
      cout << "Noto'g'ri ahrf kiritildi. Yangittan kiriting..."
```

```
    break; // eng ohirida chart emas.
 }//end switch - switch bloki tugaganligi belgisi
}//end while
cout << "\n\n\nHar bir hol uchun o'yinlar soni:"
    << "\nDurang: " << durang
    << "\nBirinchi komanda yutug'i: " << birinchi
    << "\nIkkinchi komanda yutug'i: " << ikkinchi
    << endl;
return (0);
}
```

Bu dasturda uch hil holat uchun qo'llanuvchi harflarni kiritadi. **While** takrorlash strukturasining shart berilish qismida () qavslarga olingan (natija = cin.get()) qiymat berish amali birnchi bo'lib bajariladi. cin.get() funksiyasi klaviaturadan bitta harfni o'qib oladi va uning qiymatini int tipidagi natija o'zgaruvchi-sida saqlaydi. harflar (character) odatda char tipidagi o'zgaruvchilarda saqlanadi. Lekin C++ da harflar istalgan integer (butun son) tip ichida saqlanishi mumkin, chunki kompyuter ichida harflar bir baytlik butun son tiplarida saqlanadi. Qolgan butun son tiplari esa bir baytdan kattadir. Shu sababli biz harflarni butun son (int) sifa-tida yoki harf sifatida ishlatishimiz mumkin.

```
cout << "L harfi int tipida " << static_cast<int>('L') << " ga teng." << enl;
```

Ekranda:

L harfi int tipida 76 ga teng.

Demak L harfi komputer ichida 76 qiymatiga egadir. Hozirgi kunda kompyuterlarning asosiy qismi ASCII kodirovkada ishlaydi. (American Standard Code for Information Interchange - informatsiya ayrboshlash uchun amerika standart kodi) ASCII da 256 ta belgining raqami berilgan. Bu kodirovka 8 bit - bir bayt joy oladi. Va ichida asosan lotin alofbosi harflari berilgan. Milliy alifbolarni ifodalash uchun (arab, hitoy, yahudiy, kiril) uangi kodirovka - UNICODE ishlatilmoqda. Bunda bitta simvol yki belgi ikkita bayt orqali beriladi. Ifodalanishi mumkin bo'lgan harflar soni 65536 tadir
(2 ning 16 chi darajasi). UNICODE ning asosiy noqulayligi - uning hajmidir. U asosan Internetga mo'ljallangan edi. Oldin ASCII bilan berilgan tekst hozir UNICODE da berilsa, uning hajmi ikki baravar oshib ketadi, yani aloqa tarmoqlariga ikki marta ko'proq og'irlik tushadi. Tenglashtirish ifodasining umumiy qitmati chap argumentga berilayatgan qiymatbilan tengdir. Buning qulaylik tarafi shundaki, biz deb yozishimiz mumkin. Bunda oldin g nolga

```
d = f = g = 0;
```

tenglashtiriladi keyin g = 0 ifodasining umumiy qiymati - 0 f va d larga zanjir ko'rinishida uzatilinadi. Demak, natija = cin.get() ifodasining umumiy qiymati EOF (End Of File – file ohiri) constantasi qiymati bilan solishtiriladi, va unga teng bo'lsa while takrorlash strukturasidan chiqiladi. EOF ning qiymati ko'pincha -1 bo'ladi. Lekin ANSI standarti EOF ni manfiy son sifatida belgilagan, yani uning qiymati -1 dan farqli bo'lishi mumkin. Shu sababli -1 ga emas, EOF ga tenglikni test qilish programmaning universalligini, bir sistemadan boshqasiga osonlik bilan o'tishini taminlaydi. EOF ni kiritish uchun qo'llanuvchi mahsus tugnalar kombinatsiyasini bosadi. Bu bilan u "boshqa kiritishga ma'lumot yo'q" deganday bo'ladi. EOF qiymati <iostream.h> da aniqlangan. DOS va DEC VAX VMS sistemalarida EOF ni kiritish uchun <ctrl-z> tugmalari bir vaqtda bosiladi. UNIX sistemalarida esa <ctrl-d> kiritiladi. Qo'llanuvchi harfni kiritib, ENTER (RETURN) tugmasini bosgandan so'ng, **cin.get()** funksiyasi harfni o'qiydi. Bu qiymat EOF ga teng bo'lmasa, while tanasi bajariladi. natija ning qiymati case etiketlarining qiymatlari bilan solishtiriladi. Masalan natija 'D' yoki 'd' ga teng bolda during o'zgaruvchisining qiymati bittaga oshiriladi. Keyin esa break orqali switch tanasidan chiqiladi. switch ning bir hususiyati shundaki, ifodalar bloki {} qavslarga olinishi shart emas.Blokning kirish nuqtasi case etiketi, chiqish nuqtasi esa break operatoridir.

```
 case '\n':
case '\t':
case ' ' :
   break;
```

Yuqoridagi dastur bloki qo'llanuvchi yanglish kiritgan yangi satr,tabulyatsiya va bo'shliq belgilarini filtrlash uchun yozilgan. Eng ohirgi break ning majburiy emasligi-ning sababi shuki,break dan so'ng boshqa operatorlar yo'q
Demak break qo'yilmagan taqdirda ham hech narsa bajaril-maydi.**EOF** kiritilgandan so'ng while tugaydi, o'zgaruvchi-lar ekranga bosib chiqariladi.

3. FUNKSIYALAR

C++ da dasturlashning asosiy bloklaridan biri funksiya-lardir. Funksiyalarning foydasi shundaki, katta masala bir necha kichik bo'laklarga bo'linib, har biriga alohida funksiya yozilganda, masala yechish algoritmi ancha soddalashadi. Bunda dasturchi yozgan funksiyalar C++ ning standart kutubhonasi va boshqa firmalar yozgan kutub-honalar ichidagi funksiyalar bilan birlashtiriladi. Bu esa ishni osonlashtiradi. Ko'p holda dasturda takroran
bejariladigan amalni funksiya sifatida yozish va kerakli joyda ushbu funksiyani chaqirish mumkin. Funksiyani programma tanasida ishlatish uchun u chaqiriladi, yani uning ismi yoziladi va unga kerakli argumentlar beriladi.
() qavslar ushbu funksiya chaqirig'ini ifodalaydi. Masalan:

```
 foo();
k = square(l);
```

Demak, agar funksiya argumentlar olsa, ular () qavs ichida yoziladi. Argumentsiz funksiyadan keyin esa () qavslarning o'zi qo'yiladi.

MA'LUMOTLAR TIPI (DATA TYPES)

Shu paytgacha ma'lumotlar tipi deganda butun son va kasrli son bor deb kelgan edik. Lekin bu bo'limda maylumotlar tipi tushunchasini yahshiroq ko'rib chiqish kerak bo'ladi. Chunki funksiyalar bilan ishlaganda argument kiritish va qiymat qaytarishga to'g'ri keladi. Agar boshidan boshlaydigan bo'lsak, kompyterda hamma turdagi ma'lumotlar 0 va 1 yordamida kodlanadi. Buning sababi shuki, elektr uskunalar uchun ikki holat tabiyi-dir, tok oqimi bor yoki yo'q, kondensatorda zaryad bor

yoki yo'q va hakozo. Demak biz bu holatlarni oladigan jihozlarni bir quti deb faraz qilsak, quti ichida yo narsa bo'ladi, yo narsa bo'lmaydi. Mantiqan buni biz bir yoki nol deb belgilaymiz. Bu kabi faqat ikki holatga ega bo'lishi mumkin bo'lgan maylumot birligiga biz BIT deymiz. Bu birlik kichik bo'lgani uchun kompyuterda bitlar guruhi qo'llaniladi. Bittan keyingi birlik bu BAYT (byte). Baytni sakkizta bit tashkil etadi. Demak bir bayt yordamida biz 256 ta holatni kodlashimiz mumkin bo'ladi. 256 soni ikkining sakkizinchi darajasiga tengdir. Bitimiz ikki holatga ega bo'lgani uchun biz kompyuterni ikkili arifmetikaga asoslangan deymiz. Ammo agar kerak bo'lsa, boshqa sistemaga asoslangan mashinalarni ham qo'llash mumkin. Masalan uchli sanoq sistemasiga asoslangan kompyuterlar bor. Informatika faniga ko'ra esa, hisoblash mashinasi uchun eng optimal sanoq sistemasi e ga teng bo'lar ekan. Demak amaldagi sistemalar ham shu songa iloji borisha yaqin bo'lishi kerakdir. C/C++ da baytga asoslangan tip char dir. char tipi butun son tipida bo'lib, chegaraviy qiymatlari -128 dan +127 gachadir. Bu tip lotin alifbosi harflarini va y ana qo'shimcha bir guruh simvollarni kodlashga qulay bo'lgan. Lekin hozirda milliy alifbelarni kodlash uchun 16 bitlik UNICODE qo'llanilmoqda. U yordamida 65536 ta simvolni ko'rsatish mumkin. char tipida o'zgaruvchi e'lon qilish uchun dasturda

char g, h = 3, s;

kabi yozish kerak. O'zgaruvchilar vergul bilan ayriladi. E'lon bilan bir vaqtning o'zida boshlang'ich qiymat ham berish imkoni bor. Mashina ichida baytdan tashkil topgan boshqa kattaliklar ham bor. Ikki baytdan tuzilgan kattalik so'z (word) deyiladi, unda 16 bit bo'ladi. 4 ta bayt guruhi esa ikkili so'z (double word) bo'ladi. Bu birlik 32 bitli mashimalarda qo'llaniladi. Hozirda qo'llanilmoqda bo'lgan mashinalar asosan 32 bitlidir, masalan Pentium I/II/III sistemalari. C++ da butun sonlarning ikki tipi bor. Biri char - uni ko'rib chiqdik. Ikkinchisi int dir. Mashinalarning arhitekturasi qanday kattalikda bo'lsa, int tipining ham kattakigi huddi shunday bo'ladi. 16 bitlik mashinalarda int 16 bit edi. Hozirda esa int ning uzunligi 32 bitdir. int (integer - butun son) tipi charga o'hshaydi. Farqi bir baytdan kattaligidadir. 16 bitli int ning sig'imi -32768 dan +32767 gachadir. 32 bitli int esa -2 147 483 648 dan +2 147 483 647 gacha o'rin egallaydi. Bu ikki butun son tipidan tashqari C++ da ikki tur vergulli, (nuqtali) yani haqiqiy son tipi mavjud. Bulardan biri float, hotirada 4 bayt joy egallaydi. Ikkinchisi esa double, 8 bayt kattalikka ega. Bularning harakteristikalari quyidagi jadvalda berilgan. Ushbu tiplar bilan ishlaganda unsigned(ishorasiz, +/- siz), signed (ishorali) long (uzun) va short (qisqa) sifatlarini qo'llasa bo'ladi. unsigned va signed ni faqat butun son tiplari bilan qo'llasa bo'ladi. unsigned qo'llanganda sonning ishorat biti bo'lmaydi, ishorat biti sonning kattaligini bildirish uchun qo'llaniladi. Masalan char tipida 8 chi, eng katta bir odatda ishorat bitidir. Biz unsigned char ch; desak, ch o'zgaruvchimizga faqat 0 va musbat qiymatlarni berishimiz mumkin. Lekin oddiy char [-128;127] ichida bo'lsa, unsigned char [0;255] orasidagi qiymatlarni oladi, chunki biz ishorat biti ham qo'llamoqdamiz. Huddi shunday unsigned int da (4 baytli) qiymatlar [0;4 294 467 296] orasida yotadi.

signed ni ishlatib esa biz ochiqchasiga butun sonimizning ishorati bo'lishi kerakligini bildiramiz. Normalda agar signed yoki unsigned qo'yilmasa, tipimizning ishorasi bo'ladi. **long int** bilan qo'llanilganda 16 bitli int 32 ga aylanadi. Bu agar mashina 16 bitli bo'lsa, mashina 32 bitli arhitekturaga ega bo'lsa, int ning kattaligi 4 bayligicha qolaveradi. long double tipi esa 10 bayt joy oladi. Short sifati int bilan qo'llanilganda 32 bit emas, 16 bit joy egallashga boshlaydi. Tezlikni oshirish maqsadida kam joy egallaydigan ma'lumot tiplarini qo'llash maqsadga muofiqdir. Agar tipning nomi yozilmagan bo'lsa, o'zgaruvchi int tipiga ega deb qabul qilinadi.

Ma'lumot tiplarining nomlari	Sinonimlar	Keng tarqalgan harakteristikalari
long double		10 bayt, +/-3.4e-4932...+/-3.4e4932
double		8 bayt, +/-1.7e-308...+/-1.7e308
float		4 bayt, +/-3.4e-38...+/-3.4e38
unsigned long int	unsigned long	
long int	long	
unsigned int	unsigned int	
unsigned short int	unsigned short	
short int	short	
unsigned char		
short		
char		

char va **int** dan tashqari C++ da yana bir necha integral tiplar mavjud. Bulardan biri bool tipidir. bool tipi faqat ikki farqli qiymat olishi mumkin. Bittasi true (to'g'ri) ikkinchisi false (noto'g'ri). bool tipi mantiqiy arifmetika amallarini bajarganda juda qo'l keladi. bool tipi boshqa bir integral tipga asoslangan bo'lishiga qaramasdan (int yoki char), yuqoridagi ikki qiymatdan tashqari boshqa qiymat ololmaydi. bool tipi o'zgaruvchilari to'g'ri shaklda initsalizatsiya qilinmagan taqdirda, ularning qiymati hato ravishda na true va na false bo'lishi mumkin.Yana boshqa bir integral tip bu **wchar_t** dir (wide char type - keng simvol tipi). U ham ko'pincha boshqa bir butun son tipiga asoslanadi - bir baytdan kattaroq bo'lishi kerakligi uchun short int qo'llaniladi.**wchar_t** simvollar kodlanishida qo'llaniladi. Masalan C++ da UNICODE ni odatda wchar_t bilan kodlaymiz. Hozirda wchar_t ning kattaligi 16 bit, lekin yuqori kattaligi necha bit bo'lishi kerakligi standartda belgilanmagan. Butun sonlarni C++ da bir necha asosda berish mumkin. Hech qanday belgi qo'yilmasdan yozilgan son o'nlik asosda (decimal) deb qabul qilinadi.

Sakkizli asosdagi (octal) sonni berish uchun sondan oldin 0o yoki 0O belgilarini qo'yish kerak.O'n oltilik sistema-dagi (hexadecimal) sonlar oldiga 0x yoki 0X lar yoziladi. Sakkizli sistemada qo'llaniladin raqamlar to'plami 0,1,2,3,4,5,6 va 7 dir. O'n oltilik asosda 0 dan 9 gacha sonlar, 10 - a, 11 - b, 12 - c, 13 - d, 14 - e va 15 uchun f qo'llaniladi. Harflar katta bo'lishi ham mumkin. Harflarning registorining (katta-kichikligi) farqi yo'q. Misol beraylik:

```
char d = 10, j = 0o11; // d 10 ga teng, j 9 ga teng.
int f = 0X10;          // f 16 ga teng
```

Butun son va kasr son tiplaridan tashqari C++ da void (bo'sh, hech narsa) tipi ham mavjud. Bu tipning oladigan qiymatlari bo'sh to'plamga tengdir. Void tipidagi ma'lumot chala tugallangan hisoblanadi. Boshqa turdagi ma'lumotni void ga keltirish mumkindir. Bu tip bilan ishlashni dasturlarimizda ko'rib chiqamiz.

MA'LUMOTLAR TIPINI KELTIRISH (DATA CASTING)

Gohida bir turdagi o'zgaruvchining qiymatini boshqa tipdagi o'zgaruvchiga berish kerak bo'ladi. Bu amal ma'lumot tipini keltirish (data type casting) deyiladi. Ko'p hollarda bu amal avtomatik ravishda, kompilyator tarafidan bajariladi.
Masalan ushbu parchani ko'raylik:

```
 char c = 33;
int k;
 k = c;
```

Bu yerda k ning sig'imi c nikidan kattaroqdir. Shuning uchun c ning qiymatini k ga berishda hech qanday muammo paydo bo'lmaydi. Quyidagi misolni ko'raylik:

```
 int i = 5;
float f = 9.77;
float result;
 result = f + i;
```

C++ ning kompilyatori ikki turdagi o'zgaruvchilar bilan ishlay olmaydi. Shu sababli ifodadagi sig'imi kichik bo'lgan o'zgaruvchilar ifodadagi qatnashgan eng katta sig'imga o'tqaziladi. Bu ham avtomatik tarzda bajariladi. i o'zgaruvchimiz qiymati vaqtinchalik **float** tipidagi o'zgaruvchiga beriladi. Bu vaqtinchalik o'zgaruvchi esa f ga qo'shiladi. Chiqqan javob result ga beriladi. Yuqorida ko'rib chiqqanlarimiz kompilyator tarafidan bajariladi. Bu kabi tip o'zgarishlarini avtomatik konversiya(implicit conversion) deymiz. Lekin gohida to'g'ri kelmaydigan tiplarni birga qo'llashga to'g'ri keladi. Masalan **float** tipiga double tipni o'tqazish, char ga int va hokazo. Bu hollarda ochiq konversiya (explicit conversion) amalini bajarishimiz kerak. Buni bajarishning ikki uslubi bor. Birinchisi C da qo'llaniladigan yo'l, ikkinchisi C++ uslubi. C da tipni keltirish uchun o'zgaruvchi oldiga kerakli tipni () qavslar ichida yozamiz.

```
 int k = 100;
char s;
 s = (char)k;
```

Yuqorida k ning qiymatini char tipidagi vaqtinchalik o'zgaruvchiga berildi, keyin s ga ushbu o'zgaruvchi qiymatini berildi.Bu yerda etibor berilishi kerak bo'lgan narsa shuki, 100 char ga ham to'g'ri keladi. Agar k ning qiymati char oladigan qiymattan kattaroq/kichikroq bo'ladigan bo'lsa, bu hato olib keladi. Shu sababli C dagi tip keltirish nisbatan havfli hisoblanadi. Lekin albatta bilib qo'llanilsa katta yordam beradi. C++ da ma'lumotlar tipini keltirish uchun mahsus operatorlar kiritildi. C uslubidagi keltirish hamma sharoitda qo'llanilar edi. C++ ning keltirish operatorlari esa faqat o'ziga ajratilgan funksiyalarni bajaradi. Undan tashqari ular C dagi keltirishlardan ko'ra kuchsizroqdir. Shu sababli hato ehtimoli kamaytirildi. Yana bir afzallik tarafi shundaki, yangi stildagi keltirish operatorlari tip tekshirishlarini bajarishadi, agar noto'g'ri keltirish bajarilsa, bu sintaktik hatoga olib keladi.
Ular quyida berilgan:

```
 static_cast
dynamic_cast
const_cast
reinterpret_cast
```

static_cast ni ko'rib chiqaylik.

```
 int k = 200;
char h;
 h = static_cast<char>(k);
```

static_cast dan keyin kerakli tip nomi <> qavslar ichida beriladi, va tipi o'zgarishi kerak bo'lgan o'zgaruvchi () qavslar ichida parametr sifatida beriladi. Static_cast kompilyatsiya davrida tip keltirishlarida qo'llaniladi.**dynamic_cast** esa dastur ishlash davrida tip keltirishlari uchun qo'llaniladi.
const_cast esa o'zgaruvchilardan const (o'zgarmas) va volatile (o'zgaruvchan, uchuvchan) sifatlarini olib tashlashda qo'llaniladi. Odatda const o'zgaruvchining qiymatini o'zgartirib bo'lmaydi. Ushbu holda const_cast qo'llaniladi. **reinterpret_cast** odatiy bo'lmagan keltirishlarni bajarishda qo'llaniladi (masalan void* ni int ga). reinterpret_cast o'zgaruvchining bitlarini boshqa ma'noda qo'llashga imkon beredi. Bu operator bilib ishlatilinishi kerak. Ushbu operatorlar bilan keyinroq yanada yaqin tanishamiz.

MATEMATIK KUTUBHONA FUNKSIYALARI

Standart kutubhonaning matematik funksiyalari ko'pgina amallarni bajarishga imkon beradi. Biz bu kutubhona misolida funksiyalar bilan ishlashni ko'rib chiqamiz.
Masalan bizning dasturimizda quyidagi satr bor bo'lsin:

```
 double = k;
int m = 123;
k = sin(m);
```

kompilyator uchbu satrni ko'rganida,standart kutubhonadan sin funksiyasini chaqiradi. Kirish qiymati sifatida m ni berdik. Javob, yani funksiyadan qaytgan qiymat k ga berildi.Funksiya agumentlari o'zgarmas sonlar (konstanta) o'zgaruvchilar, ifodalar va boshqa mos keluvchi qiymat qaytaradigan funksiyalar bo'lishi mumkin. Masalan:

```
int g = 49, k = 100;
cout << "4900 ning ildizi -> "<< sqrt( g * k );
```

Ekranda:

```
4900 ning ildizi -> 70;
```

Matematik funksiyalar aksariyat hollarda double tipidagi qiymat qaytarishadi.Kiruvchi argumentning tipi sifatida esa double ga keltirilishi mumkin bo'lgan tip beriladi. Bu funksiyalarni ishlatish uchun math.h (yangi ko'rinishda cmath)e'lon faylini include bilan asosiy dastur tanasiga kiritish kerak.Quyida matematik funksiya-lar kutubhonasining bazi bir a'zolarini beraylik. x va y o'zgaruvchilari double tipiga ega.

Funksiya	Aniqlanishi	Misol
ceil(x)	x ni x dan katta yoki unga teng b-n eng kichik butun songacha yahlitlaydi	ceil(12.6) = 13.0 ceil(-2.4) = -2.0
cos(x)	x ning trigonometrik kosinusi (x radianda)	cos(0.0) = 1.0
exp(x)	e ning x chi darajasi (eskponetsial f-ya)	exp(1.0) = 2.71828 exp(2.0) = 7.38906
fabs(x)	x ning absolut qiymati	x>0 => abs(x) = x x=0 => abs(x) = 0.0 x<0 => abs(x) = -x
floor(x)	x ni x dan kichik bo'lgan eng katta butun songacha yahlitlaydi	floor(4.8) = 4.0 floor(-15.9) = -16.0
fmod(x,y)	x/y ning qoldig'ini kasr son tipida beradi	fmod(7.3,1.7) = 0.5
log(x)	x ning natural lagorifmi (e asosiga ko'ra)	log(2.718282) = 1.0
log10(x)	x ning 10 asosiga ko'ra lagorifmi	log10(1000.0) = 3.0
pow(x,y)	x ning y chi darajasini beradi	pow(3,4) = 81.0 pow(16,0.25) = 2
sin(x)	x ning trigonometrik sinusi (x radianda)	sin(0.0) = 0.0
sqrt(x)	x ning kvadrat ildizi	sqrt(625.0) = 25.0
tan(x)	x ning trigonometrik tangensi (x radianda)	tan(0.0) = 0

FUNKSIYALARNING TUZILISHI

Funksiyalar dasturchi ishini juda yengillashtiradi. Funksiyalar yordamida programma modullashadi, qismlarga bo'limadi. Bu esa keyinchalik dasturni rivojlantirishni osonlashtiradi. Dastur yozilish davrida hatolarni topishni yengillashtiradi. Bir misolda funksiyaning asosiy qismlarini ko'rib chiqaylik.

```
int foo(int k, int t) {
  int result;
   result = k * t;
  return (result);
}
```

Yuqoridagi foo funksiyamizning ismi, () qavslar ichidagi parametrlar – int tipidagi k va t lar kirish argument-laridir, ular faqat ushbu funksiya ichida ko'rinadi va qo'llaniladi. Bunday o'zgaruvchilar lokal(local-mahalliy) deyiladi. result foo() ning ichida e'lon qilinganligi uchun u ham lokaldir. Demak biz funksiya ichida o'zgaruvchilarni va klaslarni (class) e'lon qilishimiz mumkin ekan. Lekin funksiya ichida boshqa funksiyani e'lon qilib bo'lmaydi. foo() funksiyamiz qiymat ham qaytaradi. Qaytish qiymatining tipi foo() ning e'lonida eng boshida kelgan - int tipiga ega. Biz funksiyadan qaytarmoqchi bo'lgan qiymatning tipi ham funksiya e'lon qilgan qaytish qiymati tipiga mos kelishi kerak - ayni o'sha tipda bo'lishi yoki o'sha tipga keltirilishi mumkin bo'lgan tipga ega bo'lishi shart. Funksiyadan qiymatni return ifodasi bilan qaytaramiz. Agar funksiya hech narsa qaytarmasa e'londa void tipini yozamiz. Yani:

```
void funk(){
  int g = 10;
   cout << g;
  return;
}
```

Bu funksiya void (bo'sh, hech narsasiz) tipidagi qiymatni qaytaradi. Boshqacha qilib aytganda qaytargan qiymati bo'sh to'plamdir. Lekin funksiya hech narsa qaytarmaydi deya olmaymiz. Chunki hech narsa qaytarmaydigan mahsus funksiyalar ham bor. Ularning qaytish qiymati belgilana-digan joyga hech narsa yozilmaydi. Biz unday funksiyalarni keyinroq qo'rib chiqamiz. Bu yerda bir nuqta shuki, agar funksiya mahsus bo'lmasa, lekin oldida qaytish qiymati tipi ko'rsatilmagan bo'lsa, qaytish qiymati int tipiga ega deb qabul qilinadi. **void** qaytish tipli funksiyalardan chiqish uchun return; deb yozsak yetarlidir.Yoki return ni qoldirib ketsak ham bo'ladi. Funksiyaning qismlari bajaradan vazifasiga ko'ra turlicha nomlanadi. Yuqorida korib chiqqanimiz funksiya aniqlanishi (function definition) deyiladi, chunki biz bunda funksiyaning bajaradigan amallarini funksiya nomidan keyin,{} qavslar ichida aniqlab yozib chiqyapmiz. Funksiya aniqlanishida {} qavslardan oldin nuqta-vergul (;) qo'yish hatodir. Bundan tashqari funksiya e'loni, prototipi yoki deklaratsiyasi (function prototype) tushunchasi qo'llaniladi. Bunda funksiyaning nomidan keyin hamon nuqta-vergul qo'yiladi, funksiya tanasi esa berilmaydi. C++ da funksiya qo'llanilishidan oldin uning aniqlanishi yoki hech bo'lmaganda e'loni kompilyatorga uchragan bo'lishi kerak. Agar funksiya e'loni boshqa funksiyalar aniqlanishidan tashqarida berilgan bo'lsa, uning kuchi ushbu fayl ohirigacha boradi. Biror bir funksiya ichida berilgan bo'lsa kuchi faqat o'cha funksiya ichida tarqaladi. E'lon fayllarda aynan shu funksiya e'lonlari berilgan bo'ladi. Funksiya e'loni va funksiya aniqlanishi bir-biriga mos tushishi kerak.

Funksiya e'loniga misol:

```
double square(char, bool);
float average(int a, int b, int c);
```

Funksiya e'lonlarda kirish parametrlarining faqat tipi yozish kifoya, huddi square() funksiyasidek. Yoki kiruvchi parametrlarning nomi ham berilishi mumkin, bu nomlar kompilyator tarafidan etiborga olinmaydi, biroq dasturning o'qilishini ancha osonlashtiradi. Bulardan tashqari C++ da funksiya imzosi (function signature) tushunchasi bor. Funksiya imzosiga funksiya nomi, kiruvchi parametrlar tipi, soni, ketma-ketligi kiradi. Funksiyadan qaytuvchi qiymat tipi imzoga kirmaydi.

```
int foo();            //No1
int foo(char, int);   //No2
double foo();         //No3 - No1 funksiya bilan imzolari ayni.
void foo(int, char);  //No4 - No2 bilan imzolari farqli.
char foo(char, int);  //No5 - No2 bilan imzolari ayni.
int foo(void);        //No6 - No1 va No3 bilan imzolari ayni,
                      //      No1 bilan e'lonlari ham ayni.
```

Yuqoridagi misolda kirish parametrlari bo'lmasa biz () qavsning ichiga void deb yozishimiz mumkin (No6 ga qarang). Yoki () qavslarning quruq o'zini yozaversak ham bo'ladi (No1 ga qarang). Yana bir tushuncha - funksiya chaqirig'idir. Dasturda funksiyani chaqirib,qo'llashimiz uchun uning chaqiriq ko'rinishini ishlatamiz. () qavslari funksiya chaqirig'ida qo'llaniladi. Agar funksiyaning kirish argumentlari bo'lmasa, () qavslar bo'sh holda qo'llaniladi. Aslida () qavslar C++ da operatorlardir. Funksiya kirish parametrlarini har birini ayri-ayri yozish kerak, masalan yuqoridagi

```
float average(int a, int b, int c);
```

funksiyasini

```
float average(int a,b,c); // Hato!
```

deb yozishimiz hatodir.

Hali etib o'tganimizdek, funksiya kirish parametrlari ushbu funksiyaning lokal o'zgaruvchilaridir. Bu o'zgaruvchilarni funksiya tanasida boshqattan e'lon qilish sintaksis hatoga olib keladi. Bir dastur yozaylik.

```
//Funksiya bilan ishlash
# include <iostream.h>
int foo(int a, int b); //Funksiya prototipi,
                       //argumentlar ismi shart emas.
int main()
{
  for (int k = 1; k <6; k++){
    for (int l = 5; l>0; l--){
      cout << foo(k,l) << " ";       //Funksiya chaqirig'i.
    }//end for (l...)
    cout << endl;
  }//end for (k...)
return (0);
} //end main()
//foo() funksiyasining aniqlanishi
int foo(int c, int d)
{ //Funksiya tanasi
  return(c * d);
}
```

Ekranda:

```
 5 4 3 2 1
10 8 6 4 2
15 12 9 6 3
20 16 12 8 4
25 20 15 10 5
```

Bizda ikki sikl ichida foo() funksiyamiz chaqirilmoqda. Funksiyaga k va l o'zgaruvchilarining nushalari uzatil-moqda. Nushalarning qiymati mos ravishda funksiyaning aniqlanishida berilgan c va d o'zgaruvchilarga berilmoqda. k va l ning nushalari deganimizda adashmadik, chunki ushbu o'zgaruvchilarining qiymatlari funksiya chaqirig'idan hech qanday ta'sir ko'rmaydi. C++ dagi funksiyalarning bir noqulay tarafi shundaki, funksiyadan faqat bitta qiymat qaytadi. Undan tashqari yuqorida ko'rganimizdek, funksiyaga berilgan o'zgaruvchilarning faqat nushalari bilan ish ko'rilarkan. Ularning qiymatini normal sharoitda funksiya ichida o'zgartirish mumkin emas. Lekin bu muammolar ko'rsatkichlar yordamida osonlikcha hal etiladi. Funksiya chaqiriqlarida avtomatik ma'lumot tipining konversiyasi bajariladi. Bu amal kompilyator tomonidan bajarilganligi sababli funksiyalarni chaqirganda ehtiyot bo'lish kerak. Javob hato ham bo'lishi mumkin. Shu sababli kirish parametrlar tipi sifatida katta hajmli tiplarni qo'llash maqsadga muofiq bo'ladi. Masalan double tipi har qanday sonli tipdagi qiymatni o'z ichiga olishi mumkin. Lekin bunday qiladigan bo'lsak, biz tezlikdan yutqazishimiz turgan gap. Avtomatik konversiyaga misol keltiraylik.

```
int division(int m, int k){
return (m / k);
}
```

dasturda chaqirsak:

```
...
float f = 14.7;
double d = 3.6;
int j = division(f,d); //f 14 bo'lib kiradi, d 3 bo'lib kiradi
                       // 14/3 - butun sonli bo'lish esa 4 javobini beradi
cout << j;
...
```

Ekranda:

Demak kompilyator f va d o'zgaruvchilarining kasr qismlarini tashlab yuborar ekan. Qiymatlarni pastroq sig'imli tiplarga o'zgartirish hatoga olib keladi.

E'LON FAYLLARI

Standart kutubhona ichidagi funksiyalarni ishlatish uchun ularning prototiplari joylashgan e'lon fayllarini include preprosessor buyrug'i bilan dastur ichiga kirgazish kerak.Quyida biz ba'zi bir keng qo'llaniladigan e'lon fayllarini keltirib o'tamiz, ularning yangi va bor bo'lsa eski ismlarini beramiz. Quyida yangi bo'lgan atamalarni keyinchalik tushuntirib o'tamiz.

<assert.h> Dastur ishlashini diagnostika qilish uchun kerakli makrolar va ma'lumotlarni e'lon qiladi. Yangi ismi <cassert>.

<ctype.h>Simvollarni test qilishda va harflar registorini kattadan kichikka va teskarisiga o'zgartirishda qo'l-laniladigan funksiyalar e'lonlarini o'z ichiga oladi. Yangi ismi <cctype>.

<float.h> Kasrli (haqiqiy) sonlarning sistemaga bog'liq limitlari aniqlangan. Yangi ismi <cfloat>.

<limits.h> Butun sonlarning sistemaga bog'liq limitlari berilgan. Yangi ismi <climits>.

<math.h> Matematik funksiyalar kutubhonasini e'lon qiladi. Yangi ismi <cmath>.

<stdio.h> Standart kirish/chiqish funksiyalarining e'lonlari berilgan. Yangi ismi <cstdio>.

<stdlib.h> Sonlarni tekstga, tekstni songa aylantiruvchi funksiyalar, hotira bilan ishlaydigan funksiyalar, tasodifiy sonlar generatsiya qiluvchi funksiyalari va boshqa yordamchi funksiyalar e'lonlarini o'z ichiga oladi. Yangi ismi <cstdlib>.

<string.h> C uslubida satrlar bilan ishlovchi funksiyalar e'loni berilgan. Yangi ismi <cstring>.

<time.h> Vaqt va sana bilan ishlaydigan funksiylar e'lonlari berilgan. Yangi ismi <ctime>.

<iostream.h>Standart kirish/chiqish oqimi bilan ishlovchi funksiyalar e'loni kiritilgan. Yangi ismi <iostream>.

<iomanip.h> Oqim manipulyatorlari berilgan. Yangi ismi <iomanip>.<fstream.h> Diskda joylashgan fayllar bilan kirish/chiqish amallarini bajaruvchi funksiyalar ellonlari berilgan. Yangi ismi <fstream>. Quyidagi e'lon fayllarining faqat bitta ismi bir.

<utility> Boshqa kutubhonalar tomonidan qo'llaniladigan yordamchi funksiyalar va klaslarning e'lonlari kiritilgan

<vector>, <list>, <deque>, <queue>, <stack>, <map>, <set>, <bitset> standart kutubhona konteyner klaslarining e'lonlarini o'z ichiga olganlar.

<functional> Standart kutubhona algoritmlari tomonidan qo'llaniladigan klas va funksiyalarini e'lon qiladi.

<memory> Standart kutubhona konteynerlari uchun hotira ajratishda qo'llaniladigan funksiya va klaslar e'lonlari berilgan.

<iterator> Konteynerlar ichidagi ma'lumotlar manipulyatsiyasida qo'llaniladigan iterator klaslari e'lonlari berilgan.

<algorithm> Konteynerlardagi ma'lumitlarni qayta ishlashda qo'llaniladigan funksiya va klaslar e'lonlari berilgan.

<exception>, <stdexcept> Fafqulotda hodisalar mehanizmini bajaruvchi klaslar berilgan.

<string> Standart kutubhonaning string klasi e'loni berilgan.

<sstream> Hotiradagi satrlarga kirish/chiqishni bajaradi-gan funksiyalar prototipi berilgan.

<locale> Mahalliy sharoitga moslashgan birliklar (pul, vaqt, sonlarning turli ko'rinishlari) bilan ishlaydigan funksiyalar e'lonlari berilgan.

<limits> Hisoblash sistemalarida sonli ma'lumot tiplari-ning chegaralarini belgilashda ishlatiladigan klas e'lonlari berilgan.

<typeinfo> Ijro vaqtida tip identefikatsiyasi uchun qo'llaniladigan klaslar e'loni kiritilgan. Qo'llanuvchi yozgan e'lon fayllari .h bilan tugasa maqsadga muofiq bo'ladi. Bunday fayllar qo'shtirnoqlarga olingan holda dasturga kiritiladi, yani masalan:

```
# include "mening_faylim.h"
```

TASODIFIY QIYMATLARNI KELTIRIB CHIQARISH

Kompyuter faning ko'p sohalarida tasodifiy qiymatlar bilan ishlash kerak bo'ladi. Masalan o'yin tuzganda, ma'lum bir tabiiy hodisani modellashtirganda va hokazo. Bunda dasturlash tasodifiy qiymatni olishga asoslangan. Va o'sha tasodifiy qiymatga bog'langan holda biror bir funksiya bajariladi yoki ifoda hisoblanadi. Tasodifiy qiymatni standart kutubhonaning rand() funksiyasi bilan olsa bo'ladi. rand() 0...RAND_NAX orasida yotgan butun son qiymatini qaytaradi. RAND_MAX <stdlib.h> da aniqlangan simvolik konstantadir. Uning kattaligi 32767 dan kichik bo'lmasligi kerak. rand() qaytaradigan qiymat aslini olganda psevdo-tasodifiydir. Yani dastur qayta-qayta ishlatilganda ayni qiymatlarni beraveradi. To'la tasodifiy qilish uchun dastur har gal ishlatilganda ma'lum bir o'zgaruvchan qiymatga asoslanib boshlanishi kerak.Buning uchun srand() funksiyasi qo'llaniladi. srand() dasturda faqat bir marta chaqirilsa yetarlidir. Dastur ishlash davomida esa ehtiyojga qarab, rand() chaqirilaveradi. Tasodifiy qiymatlar bilan ishlash uchun <stdlib.h> ni include bilan e'lon qilishimiz kerak. Yuqoridagi funksiyalarning prototiplarini berib o'taylik:

```
void srand(unsigned int seed);
int rand();
```

Biz har gal dastur ishlaganda srand() ga seed o'zgaruvchisini klaviaturadan kiritishimiz mumkin, yoki buni avtomatik tarzda bo'ladigan qilishimiz mumkin.

Buning bir yo'li

```
srand( time(NULL) );
```

deb yozishdir. Bunda kompyuter avtomatik ravishda o'z ichidagi soatning qiymatini time() funksiyasi yordamida o'qiydi va srand() ga parametr sifatida beradi. time() ni NULL yoki 0 argument bilan chaqirganimizda kompyuter

soatining joriy vaqti sekundlar ko'rinishida qaytaradi. Vaqt bilan ishlash uchun <time.h> ni e'lon qilish kerak bo'ladi. Nardaning 6 lik toshidek ishlaydigan dastur yozaylik.

```
//rand()/srand() bilan ishlash.
# include <iostream.h>
# include <stdlib.h>
# include <time.h>
int main()
{
  srand( time(NULL) );
  int k;
  for (k = 1; k<11; k++){
    cout << (1 + rand() % 6) << ( (k % 5 == 0) ? endl : " " );
  }//end for(k...)
  return (0);
}//end main()
```

Bu yerda amallar siklda bajarilmoqda. rand() % 6 ifodasi masshtablash (scaling) deyiladi. rand() dan qaytuvchi qiymat 0...RAND_MAX orasida bo'lishi mumkin, biz bu qiymatni 6 ga ko'ra modulini olsak, bizda javob sifatida faqat 0, 1, 2, 3, 4 va 5 qiymatlari bo'lishi mumkin. Keyin esa biz bu qiymatlarni siljitamiz, yani 1 raqamini qo'shamiz. Shunda javoblar bizga kerakli oraliqda yotgan bo'ladi. ?: shart operatori bilan biz har bir raqamdan keyin ekranga nima bosilib chiqishini hal qilyapmiz, bu yoki endl bo'ladi, yoki bo'shliq bo'ladi. Tasodifiy qiymatlar bilan biz keyingi dasturlarimizda hali yana ishlaymiz. C/C++ dagi enumeration (ketma-ketlik) tipini tanishtirib o'taylik. Enumeration butun sonli konstantalar yig'indi-sidir.Konstantalar qiymati agar boshlangich qiymat beril- magan bo'lsa, noldan boshlanadi va 1 ga oshib boraveradi, yoki biz qiymatlarni o'zimiz berishimiz mumkin. Masalan:

```
enum MeningSinifim {AKMAL, TOHIR, NIGORA};
MeninigSinfim SinfA;
SinfA = TOHIR;
SinfA = AKMAL;
```

Bu yerdagi MeningSinfim qo'llanuvchi tarafidan tuzilgan yangi tipdir.Keyin esa ushbu tipda bo'lgan SinfA o'zgaruv chisini e'lon qilamiz. Ushbu tipdagi o'zgaruvchi faqat uchta farqli qiymatni qabul qiladi. Bunda AKMAL 0 ga teng, TOHIR 1 ga va NIGORA 2 ga teng bo'ladi. Lekin biz enumeration tipida bo'lgan o'zgaruvchilarga faqat {} qavslar ichida berilgan simvolik ismlarni bera olamiz. Yani to'g'ridan-to'g'ri sonli qiymat bera olmaymiz. Boshqa bir ketma-ketlik:

```
enum Oylar {YAN = 1, FEV, MAR, APR, MAY, IYUN, IYUL, AVG, SEN, OKT, NOY, DEK};
```

Bunda qiymatlar birdan boshlanadi va birga ortib boradi. Enumeration qiymatlari katta harflar bilan yozilsa, dastur ichida ajralib turadi. Va sonli konstantalardan ko'ra ishlash qulayroqdir.

DASTUR BIRLIKLARINING SIFATLARI

O'zgaruvchilarning kattaligi, ismi va turidan tashqari yana bir necha boshqa hossalari bor. Bulardan biri hotirada saqlanish tipidir. O'zgaruvchilar hotirada ikki uslubda saqlanishi mumkin. Birinchisi avtomatik, ikkinchisi statik yo'ldir. Avtomatik bo'lgan birlik u e'lon qilingan blok bajarilishi boshlanganda tuziladi, va ushbu blok tugaganda buziladi, u hotirada egallagan joy esa bo'shatiladi. Faqat o'zgaruvchilar avtomatik bolishi mumkin. Avtomatik sifatini berish uchun o'zgaruvchi boshiga auto yoki register so'zlari qo'yiladi. Aslida lokal o'zgaruvchilar oldiga hech narsa yozilmasa, ularga auto sifati beriladi. Dastur ijro etilganda o'zgaruvchilar markaziy prosessor registrlariga yuklanib ishlov ko'radilar. Keyin esa yana hotiraga qaytariladilar. Agar register sifatini qo'llasak, biz kompyuterga ushbu o'zgaruvchini ishlov ko'rish payti davomida registrlarning birida saqlashni tavsiya etgan bo'lamiz. Bunda hotiraga va hotiradan yuklashga vaqt ketmaydi. Albatta bu juda katta vaqt yutug'i bermasligi mumkin, lekin agar sikl ichida ishlatilsa, yutuq sezilarli darajada bo'lishi mumkin. Shuni etish kerakki, hozirgi kundagi kompilyatorlar bunday ko'p ishlatiladigan o'zgaruvchilarni ajrata olishdi va o'zlari ular bilan ishlashni optimizatsiya qilishadi. Shu sababli o'zgaruvchini register deb e'lon qilish shart bo'lmay qoldi. Hotirada boshqa tur saqlanish yo'li bu statik saqlanishdir. Statik sifatini o'zgaruvchi va funksiyalar olishlari mumkin. Bunday birliklar dastur boshlanish nuqtasida hotirada quriladilar va dastur tugashiga qadar saqlanib turadilar. O'zgaruvchi va funksiyalarni statik qilib e'lon qilish uchun static yoki extern (tashqi) ifodalari e'lon boshiga qo'yiladi. Statik o'zgaruvchilar dastur boshida hotirada quriladilar va initsalizatsiya qilinadilar. Fuksiyalarning ismi esa dastur boshidan bor bo'ladi. Lekin statik birliklar dastur boshidan mavjud bo'lishi, ularni dasturning istalgan nuqtasida turib qo'llasa bo'ladi degan gap emas. Hotirada saqlanish uslubi bilan qo'llanilish sohasi tushunchalari farqli narsalardir. O'zgaruvchi mavjud bo'lishi mumkin, biroq ijro ko'rayatgan blok ichida ko'rinmasligi mumkin. Dasturda ikki hil statik birliklar bor. Birinchi hili bu tashqi identefikatorlardir. Bular global sohada aniqlangan o'zgaruvchi va funksiyalardir. Ikkinchi tur statik birliklar esa static ifodasi bilan e'lon qilingan lokal o'zgaruvchilardir. Global o'zgaruvchi va funksiyalar oldida extern deb yozilmasa ham ular extern sifatiga ega bo'ladilar. Global o'zgaruvchilar ularning e'lonlarini funksiyalar tashqarisida yozish bilan olinadi. Bunday o'zgaruvchi va funksiyalar o'zlaridan faylda keyin keluvchi har qanday funksiya tomonidan qo'llanilishi mumkin. Global o'zgaruvchilarni ehtiyotorlik bilan ishlatish kerak. Bunday o'zgaruvchilarni harqanday funksiya o'zgartirish imkoniga ega. O'zgaruvchiga aloqasi yo'q funksiya uning qiymatini bilib-bilmasdan o'zgartirsa, dastur mantig'i buzilishi mumkin. Shu sababli global sohada iloji boricha kamroq o'zgaruvchi aniqlanishi lozim. Faqat bir joyda ishlatilinadigan o'zgaruvchilar o'sha blok ichida aniqlanishi kerak. Ularni global qilish noto'g'ridir. Lokal o'zgaruvchilarni, yani funksiya ichida e'lon qilingan o'zgaruvchilarni static so'zi bilan e'lon qilish mumkin. Bunda ular ikkinchi hil statik birliklarni tashkil qilishgan bo'lishadi. Albatta ular faqat o'sha funksiya ichida qo'llanishlari mumkin. Ammo funksiya bajarilib tugaganidan so'ng statik o'zgaruvchilar o'z qiymatlarini saqlab qoladilar va keyingi funksiya chaqirig'ida saqlanib qolingan qiymatni yana ishlatishlari yoki o'zgartirishlari mumkin. Statik o'zgaruvchilar e'lon paytida initsalizatsiya qilinadilar. Agar ularga e'lon paytida ochiqchasiga qiymat berilmagan bo'lsa, ular nolga tenglashtiriladi.

```
static double d = 0.7; // ochiqchasiga qiymat berish,
```

```
static int k;          // qiymati nol bo'ladi.
```

Agar static yoki extern ifodalari global identefikatorlar bilan qo'llanilsa, ushbu identefikatorlar mahsus ma'noga egadirlar. Biz u hollarni keyin ko'rib o'tamiz.

QO'LLANILISH SOHASI (SCOPE RULES)

O'zgaruvchi dasturning faqat ma'lum sohasida ma'moga egadir. Yani faqat biror bir blok, yoki bu blok ichida joylashgan bloklar ichida qo'llanilishi mumkin. Bunday blokni soha (qo'llanilish sohasi - scope) deb ataylik. Identefikator (oz'garuvchi yoki funksiya ismi) besh hil sohada aniqlanishi mumkin. Bular funksiya sohasi, fayl sohasi, blok sohasi, funksiya prototipi sohasi va klas sohasi. Agar identefikator e'loni hech bir funksiya ichida joylashmagan bo'lsa, u fayl sohasiga egadir. Ushbu identefikator e'lon nuqtasidan to fayl ohirigacha ko'rinadi. Global o'zgaruvchilar, funksiya prototiplari va aniqlanishlari shunday sohaga egadirlar. Etiketlar (label), yani identefikatorlardan keyin ikki nuqta (:) keluvchi ismlar, masalan: chiqish: mahsus ismlardir. Ular dastur nuqtasini belgilab turadilar. Dasturning boshqa yeridan esa ushbu nuqtaga sakrashni (jump) bajarish mumkin. Va faqat etiketlar funksiya sohasiga egadirlar. Etiketlarga ular e'lon qilingan funksiyaning istalgan joyidan murojaat qilish mumkin. Lekin funksiya tashqarisidan ularga ishora qilish ta'qiqlanadi. Shu sababli ularning qo'llanilish sohasi funksiyadir. Etiketlar switch va goto ifodalarida ishlatilinadi. goto qo'llanilgan bir blokni misol qilaylik.

```
int factorial(int k) {
if (k<2)
  goto end;
else
  return ( k*factorial(k-1) );
 end:
  return (1);
}
```

Bu funksiya sonning faktorialini hisoblaydi. Bunda 0 va 1 sonlari uchun faktorial 1 ga teng, 1 dan katta x soni uchun esa x!=x*(x-1)*(x-2)...2*1 formulasi bo'yicha hisoblanadi. Yuqoridagi funksiya rekursiya metodini ishlatmoqda, yani o'zini-o'zini chaqirmoqda. Bu usul dasturlashda keng qo'llaniladi. Funksiyamiz ichida bitta dona etiket - end: qollanilmoqda. Etiketlarni qo'llash strukturali dasturlashga to'g'ri kelmaydi, shu sababli ularni ishlatmaslikga harakat qilish kerak. Blok ichida e'lon qilingan identefikator blok sohasiga egadir. Bu soha o'zgaruvchi e'lonidan boshlanadi va } qavsda (blokni yopuvchi qavs) tugaydi. Funksiyaning lokal o'zgaruvchilari hamda funksiyaning kiruvchi parametrlari blok sohasiga egadirlar. Bunda parametrlar ham funksiyaning lokal o'zgaruvchilari qatoriga kiradilar. Bloklar bir-birining ichida joylashgan bo'lishi mumkin. Agar tashqi blokda ham, ichki blokda ham ayni ismli identefikator mavjud bo'lsa, dastur isjrosi ichki blokda sodir bo'layatgan bir vaqtda ichki identefikator tashqi blokdagi identefikatorni to'sib turadi. Yani ichki blokda tashqi blok identefikatorining ismi ko'rinmaydi. Bunda ichki blok faqat o'zining o'zgaruvchisi bilan ish yuritishi mumkin. Ayni ismli tashqi blok identefikatorini ko'rmaydi. Lokal o'zgaruvchilar static deya belgilanishlariga qaramay, faqat aniqlangan bloklaridagina qo'llanila oladilar. Ular dasturning butun hayoti davomida mavjud bo'lishlari ularning qo'llanilish sohalariga ta'sir ko'rsatmaydi. Funksiya prototipi sohasiga ega o'zgaruvchilar funksiya e'lonida berilgan identefikatorlardir. Aytib o'tkanimizdek, funksiya prototipida faqat o'zgaruvchi tipini bersak yetarlidir. identefikator ismi berilsa, ushbu ism kompilyator tomonidan hisobga olinmaydi. Bu ismlarni dasturning boshqa yerida hech bir qiyinchiliksiz qo'llash mumkin. Kompilyator hato bermaydi. Klas sohasiga ega ismlar klas nomli bloklarda aniqlanadilar. Bizlar klaslarni keyinroq o'tamiz. Hozir soha va hotirada saqlanish tipi mavzusida bir misol keltiraylik.

```
//Qo'llanilish sohasi, static va auto
//o'zgaruvchilarga misollar.
 # include <iostream.h>
 long r = 100; //global o'zgaruvchi,
              //funksiyalar tashqarisida aniqlangan
 void staticLocal(); //funksiya prototipi yoki e'loni
void globalAuto(int k /* k funksiya prototipi
              sohasiga ega */); //f-ya e'loni
 int main ()
{
   staticLocal();
   staticLocal();
    int m = 6;
   globalAuto(m);
   ::r = ::r + 30;
   cout "main da global long r: ";
   cout << ::r << endl; //global long r to'liq aniqlangan
                      //ismi o'rqali qo'llanilmoqda
   m++;//m = 7
   globalAuto(m);
    int r = 10; //tashqi sohadagi main ga nisbatan lokal o'zgaruvchi;
              //long r ni to'sadi
   cout << "tashqi sohadagi lokal r: " << r << endl;
     { //ichki blok
        short r = 3; //ichki sohadagi lokal o'zgaruvchi;
                    //int r ni to'sadi
        cout << "ichki sohadagi lokal r: " << r << endl;
```

```
    }
  cout << "tashqi sohadagi lokal r: " << r << endl;
  return (0);
}
void staticLocal() {
  static  int s = 0; //statik o'zgaruvchi
   cout << "staticLocal da: " << s << endl;
  s++; //s = 1;
}
void globalAuto(int i) {
  int g = 333; //avtomatik o'zgaruvchi
   cout << "globalAuto da: " << i << " ";
  cout << g << " ";
  g++;
  cout << r << endl; //global long r ekranga bosiladi
}
```

Ekranda:

```
staticLocal da: 0
staticLocal da: 1
globalAuto da: 6 333 100
main da global long r: 130
globalAuto da: 7 333 130
tashqi sohadagi lokal r: 10
ichki sohadagi lokal r: 3
tashqi sohadagi lokal r: 10
```

ARGUMENT OLMAYDIGAN FUNKSIYALAR

Agar funksiya prototipida () qavslar ichiga void deb yozilsa, yoki hech narsa yozilmasa, ushbu funksiya kirish argument olmaydi. Bu qonun C++ da o'rinlidir. Lekin C da bo'sh argument belgisi, yani () qavslar boshqa ma'no beradi. Bu e'lon funksiya istalgancha argument olishi mumkin deganidir. Shu sababli C da yozilgan eski dasturlar C++ kompilyatorlarida hato berishlari mumkindir. Bundan tashqari funksiya prototipi ahamiyati haqida yozib o'taylik. Iloji boricha har doim funksiya prototiplarini berib o'tish kerak, bu modulli dasturlashning asosidir. Prototip va e'lonlar alohida e'lon fayllar ichida berilishi mumkin. Funksiya yoki klas o'zgartirilganda e'lon fayllari o'zgarishsiz qoladi. Faqat funksiya aniqlangan fayllar ichiga o'zgartirishlar kiritiladi. Bu esa juda qulaydir.

inline SIFATLI FUNKSIYALAR

Funksiyalar dastur yozishda katta qulayliklar beradilar. Lekin mashina saviyasida funksiyani har gal chaqirtirish qo'shimcha ish bajarilishiga olib keladi. Registrlardagi o'zgaruvchilar o'zgartiriladi, lokal yozgaruvchilar quriladi, parametr sifatida berilgan argumentlar funksiya stekiga o'ziladi. Bu, albatta, qo'shimcha vaqt oladi. Umuman aytganda, hech funksiyasiz yozilgan dastur, yani hamma amallari faqat main() funksiyasi ichida bajariladigan monolit programma, bir necha funksiyalarga ega, ayni ishni bajaradigan dasturdan tezroq ishlaydi. Funksiyalarning bu noqulayligini tuzatish uchun inline (satr ichida) ifodasi funksiya e'loni bilan birga qo'llaniladi. inline sifatli funksiyalar tanasi dasturdagi ushbu funksiya chaqirig'i uchragan joyga qo'yiladi. inline deb ayniqsa kichik funksiyalarni belgilash effektivdir. inline ning o'ziga yarasha kamchiligi ham bor, inline qo'llanilganda dastur hajmi oshdi. Agar funksiya katta bo'lsa va dastur ichida ko'p marotaba chaqirilsa, programma hajmi juda kattalashib ketishi mumkin. Oddiy, inline sifati qo'llanilmagan funksiyalar chaqirilish mehanizmi quyidagicha bo'ladi. Dastur kodining ma'lum bir yerida funksiya tanasi bir marotaba aniqlangan bo'ladi. Funksiya chaqirig'i uchragan yerda funksiya joylashgan yerga ko'rsatkich qo'yiladi. Demak, funksiya chaqirig'ida dastur funksiyaga sakrashni bajaradi. Funksiya o'z ishini bajarib bo'lgandan keyin dastur ishlashi yana sakrash joyiga qaytadi. Bu dastur hajmini ihchamlikda saqlaydi, lekin funksiya chaqiriqlari vaqt oladi. Kompilyator inline ifodasini inobatga olmasligi mumkin, yani funksiya oddiy holda kompilyatsiya qilinishi mumkin. Va ko'pincha shunday bo'ladi ham. Amalda faqat juda kichik funksiyalar inline deya kompilyatsiya qilinadi. inline sifatli funksiyalarga o'zgartirishlar kiritilganda ularni ishlatgan boshqa dastur bloklari ham qaytadan kompilyatsiya qilinishi kerak. Agar katta proyektlar ustida ish bajarilayatgan bo'lsa, bu ko'p vaqt olishi mumkin. inline funksiyalar C da qo'llanilgan # define makrolari o'rnida qo'llanilish uchun mo'ljallangan. Makrolar emas, balki inline funksiyalar qo'llanilishi dastur yozilishini tartibga soladi. Makro funksiyalarni keyinroq o'tamiz.

```
//inline ifodasining qo'llanilishi
# include <iostream.h>
inline int sum(int a, int b);//funksiya prototipi
int main() {
  int j = -356,
      i =  490;
   cout << "j + i = " << sum(j,i) <<endl;
   return (0);
}
int sum(int a, int b){ //funksiya aniqlanishi
  return( a + b );
}
```

Ekranda:

```
j + i = 134
```

KO'RSATKICHLAR VA FUNKSIYA CHAQIRIQLARIDA ULARNING QO'LLANILISHI

C/C++ da funksiya chaqirig'iga kirish parametrlarini berishning ikki usuli bordir. Birinchi usul qiyamat bo'yicha chaqiriq (call-by-value) deyiladi. Ikkinchi usul ko'rsatkich bo'yicha chaqiriq (call-by-reference) deb nomlanadi. Hozirgacha yozgan hamma funksiyalar qiymat bo'yicha chaqirilardi. Buning ma'nosi shuki, funksiyaga o'zgaruvchining o'zi emas, balki uning nushasi argument sifatida beriladi. Buning afzal tomoni shundaki, o'zgaruvchi qiymatini funksiya ichida o'zgartirish imkoni yo'qdir. Bu esa havfsizlikni ta'minlaydi. Ammo, agar o'zgaruvchi yoki ifoda hotirada katta joy egallasa, uning nushasini olish va funksiyaga argument sifatida berish sezilarli vaqt olishi mumkin. Ko'rsatkich bo'yicha chaqiriqda o'zgaruvchi nushasi emas, uning o'zi argument sifatida funksiyaga uzatilinadi. Bu chaqiriqni bajarishning ikki usuli mavjud. Bittasini biz hozir ko'rib chiqamiz, ikkinchi usulni esa keyinroq. Hozir o'tadigan ko'rsatkichni o'zbekchada &-ko'rsatkich (AND ko'rsatkich - reference) deb ataylik. Ikkinchi tur ko'rsatkichning esa inglizcha nomlanishini saqlab qo'laylik, yani pointer (ko'rsatkich) deb nomlaylik. Bu kelishishdan maqsad, inglizchadagi reference va pointer so'zlar o'zbekchaga ko'rsatkich deb tarjima qilinadi. Bu ikki ifodaning tagida yotuvchi mehanizmlar o'zhshash bo'lishlariga qaramay, ularning qo'llanishlari farqlidir. Shuning uchun ularning nomlanishlarida chalkashlik vujudga kelmasligi kerak. Aytkanimizdek, ko'rsatkich bo'yicha chaqiriqda o'zgaruvchining o'zi parametr bo'lib funksiyaga beriladi. Funksiya ichida o'zgaruvchi qiymati o'zgartirilishi mumkin. Bu esa havfsizlik muammosini keltirb chiqarishi mumkin. Chunki aloqasi yo'q funksiya ham o'zgaruvchiga yangi qiyamat berishi mumkin. Bu esa dastur mantig'i buzilishiga olib keladi. Lekin, albatta, bilib ishlatilsa, ko'rsatkichli chaqiriq katta foyda keltirishi mumkin. &-ko'rsatkichli parametrni belgilash uchun funksiya prototipi va aniqlanishida parametr tipidan keyin & belgisi qo'yiladi. Funksiya chaqirig'i oddiy funksiyaning ko'rinishiga egadir. Pointerlarni qo'llaganimizda esa funksiya chaqirig'i ham boshqacha ko'rinishga egadir. Ammo pointerlarni keyinroq ko'rib chiqamiz. Bir misol keltiraylik.

```
//Qiymat va &-ko'rsatkichli chaqiriqlarga misol
 # include <iostream.h>
 int qiymat_10(int);            //e'lon
int korsatkich_10(int &);      //e'lon
 int f, g;
 int  main(){
   f = g = 7;
    cout << f << endl;
   cout << qiymat_10(f) << endl;
   cout << f << endl << endl;
    cout << g << endl;
   cout << korsatkich_10(g) << endl; //chaqiriq ko'rinishi o'zgarmaydi
   cout << g << endl;
    return (0);
}
 int qiymat_10(int k){
   return ( k * 10 );
}
 int korsatkich_10(int &t){
   return ( t * 100 );
}
```

Ekranda:

```
 7
70
7
7
700
700
```

Bu yerda g o'zgaruvchimiz korsatkich_10(int &) funksiyamizga kirib chiqqandan so'ng qiymati o'zgardi. Ko'rsatkich bo'yicha chaqiriqda kirish argumentlaridan nusha olinmaydi, shu sababli funksiya chaqirig'i ham juda tez bajariladi. &-ko'rsatkichlarni huddi oddiy o'zgaruvchilarning ikkinchi ismi deb qarashimiz mumkin. Ularning birinchi qo'llanilish yo'lini - funksiya kirish parametrida ishlatilishini ko'rib chiqdik. &-ko'rsatkichni blok ichida ham ko'llasak bo'ladi. Bunda bir muhim marsani unutmaslik kerakki &-ko'rsatkich e'lon vaqtida initsalizatsiya qilinishi kerak, yani ayni tipda bo'lgan boshqa bir oddiy o'zgaruvchi unga tenglashtirilishi kerak. Buni va boshqa tushunchalarni misolda ko'rib chiqaylik.

```
//const ifodasi bilan tanishish;
//&-ko'rsatkichlarning ikkinchi qo'llanilish usuli
 # include <iostream.h>
 void printInt(const int &); //funksiya prototipi
 double d = 3.999;
int j = 10;
 int main()
{
   double &rd = d; //d ga rd nomli &-ko'rsatkich
   const int &crj = j; //const ko'rsatkich
   const short int k = 3; //const o'zgaruvchi - konstanta
    cout << rd << endl;
   printInt(j);
   printInt(crj);
```

```
  return (0);
}
 void printInt(const int &i) //...int& i... deb yozish ham mumkin;
{                        //kirish argumenti const dir
  cout << i << endl;
  return;
}
```
Ekranda:
```
 3.999
10
```
Ko'rganimizdek, rd ko'rsatkichimiz d o'zgaruvchi qiymatini ekranga bosib chiqarish imkonini beradi. Ko'rsatkich o'rqali o'zgaruchining qiymatini ham o'zgartirsa bo'ladi. &-ko'rsatkichning asosiy hususiyati shundaki mahsus sintaksis - & belgisining qo'yilishi faqat ko'rsatkich e'lonida qo'llaniladi halos. Dastur davomida esa oddiy o'zgaruvchi kabi ishlov ko'raveradi. Bu juda qulaydir, albatta. Lekin ko'rsatkich ishlatilganda ehtiyot bo'lish kerak, chunki, masalan, funksiya tanasi ichida argumentning nushasi bilan emas, uning o'zi bilan ish bajarilayatganligi esdan chiqishi mumkin. const (o'zgarmas) ifodasiga kelaylik. Umuman olganda bu nisbatan yangi ifodadir. Masalan C da ishlagan dasturchilar uni ishlatishmaydi ham. const ni qo'llashdan maqsad, ma'lum bir identefikatorni o'zgarmas holga keltirishdir. Masalan, o'zgaruvchi yoki argument const bilan sifatlantirilsa, ularning qiymatini o'zgartirish mumkin emas. Lekin boshqa amallarni bajarish mumkin. Ularni ekranga chiqarish, qiymatlarini boshqa o'zgaruvchiga berish ta'qiqlanmaydi. const ifodasisiz ham dasturlash mumkin, ammo const yordamida tartibli, chiroyli va eng muhimi kam hatoli dasturlashni amalga oshirsa bo'ladi. const ning ta'siri shundaki, const qilingan o'zgaruvchilar kerakmas joyda o'zgarolmaydilar. Agar funksiya argumenti const deb belgilangan bo'lsa, ushbu argumentni funksiya tanasida o'zgartirilishga harakat qilinsa, kompilyator hato beradi. Bu esa o'zgaruvchining himoyasini oshirgan bo'ladi. *const* ning qo'llanilish shakli ko'pdir. Shulardan asosiylarini ko'rib chiqsak. Yuqoridagi misoldagi *const int &crj = j;* amali bilan biz &-ko'rsatkichni e'lon va initsalizatsiya qildik. Ammo crj ko'rsatkichimiz const qilib belgilandan, bu degani crj ko'rsatkichi orqali j o'zgaruvchisining qiymatini o'zgartira olmaymiz. Agar const ifodasi qo'llanilmaganda edi, masalan yuqoridagi ***double &rd = d;*** ifodadagi rd ko'rsatkichi yordamida d ning qiymatini qiyinchiliksiz o'zgartirishimiz mumkin. d ning qiymatini 1 ga oshirish uchun ***d++;*** yoki ***rd++;*** deb yozishimiz kifoyadir. Yana bir marta qaytarib o'taylikki, &-ko'rsatkichlar e'lon vaqtida initsalizatsiya qilinishi shartdir. Yani quyidagi ko'rinishdagi bir misol hatodir:
```
 int h = 4;
int &k; // hato!
k = h;  // bu ikki satr birga qo'shilib yoziladi: int &k = h;
```

Ba'zi bir dasturchilar &-ko'rsatkich e'londa & belgisini o'zgaruvchi tipi bilan birga yozadilar. Bunining sababi shuki, & belgisining C/C++ dagi ikkinchi vazifasi o'zgaruvchi yoki ob'ektning adresini qaytarishdir. Unda & ni o'zgaruvchiga yopishtirib yozish shartdir. Demak, & tipga yopishtirib yozish & ning qo'llanishlarini bir-biridan farqlab turadi. Lekin & ni ko'rsatkich e'lonida qo'llaganda, uning qanday yozilishining ahamiyati yo'q. Adres olish amalini biz keyinroq ko'rib chiqamiz.
```
 ...
int H = 4;
int F;
 int &k = H;
int& d = F;            // yuqoridagi e'lon bilan aynidir
 void foo(long &l);
void hoo(double& f);    // yuqoridagi protopip bilan aynidir
...
```

Bir necha ko'rsatkichni e'lon qilish uchun:
```
 int a, b , c;
int &t = a, &u = b, &s = c;
```

Bu yerda & operatori har bir o'zgaruvchi oldida yozilishi shart. &-ko'rsatkich ko'rsatayotdan hotira sohasining adresini o'zgartirib bo'lmaydi.

Masalan:
```
 int K = 6;
int &rK = K;
```

K ning hotiradagi adresi 34AD3 bolsin, rK ko'rsatkichning adresi esa 85AB4. K ning qiymati 6, rK ning qiymati ham 6 bo'ladi. Biz rK ga qaytadan boshqa o'zgaruvchini tenglashtirsak, rK yangi o'zgaruvchiga ko'rsatmaydi, balki yangi o'zgaruvchining qiymatini K ning adresiga yozib qo'yadi.

Masalan yangi o'zgaruvchi N bo'lsin, uning adresi 456F2, qiymati esa 3 bo'lsin. Agar biz
```
 rK = N;
```
desak, rK N ga ko'rsatmaydi, balki K ning adresi - 34AD3 bo'yicha N ning
qiymatini - 3 ni yozadi. Yani K ning qiymati 6 dan 3 ga o'zgaradi.
Yuqoridagi dasturda:
```
 const short int k = 3;
```
deb yozdik. Bu const ning ikkinchi usulda qo'llanilishidir. Agar o'zgaruvchi tipi oldidan const qo'llanilsa, o'zgaruvchi konstantaga aylanadi, dastur davomida biz uning qiymatini o'zgartira olmaymiz. Bu usul bilan biz Pascaldagi kabi konstantalarni e'lon qilishimiz mumkin. Bu yerda bitta shart bor, const bilan sifatlantirilgan o'zgaruvchilar e'lon davrida

initsalizatsiya qilinishlari shart. Umuman olganda bu qonun const ning boshqa joylarda qo'llanilganida ham o'z kuchini saqlaydi. Albatta, faqat funksiya argumentlari bilan qo'llanilganda bu qonun ishlamaydi. C/C++ da yana # define ifodasi yordamida ham simvolik konstantalarni e'lon qilish mumkin. Ushbu usulni keyin ko'rib chiqamiz. Undan tashqari enum strukturalari ham sonli kostantalarni belgilaydi.

Dasturimizda printInt() funksiyamizga kiradigan int& tipidagi argumentimizni const deya belgiladik. Buning ma'nosi shuki, funksiya ichida ushbu argument o'zgartirilishga harakat qilinsa, kompilyator hato beradi. Yani funksiya const bo'lib kirgan argumentlarni (hoh ular o'zgaruvchi nushalari bo'lsin, hoh o'zgaruvchilarga ko'rsatkich yoki pointer bo'lsin) qiymatlarini o'zgartira olmaydilar.

Funksiya faqat bitta qiymat qaytaradi dedik. Bu qaytgan qiymatni biz o'zgaruvchiga berishimiz mumkin. Agar bittadan ko'proq o'zgaruvchini o'zgartirmoqchi bo'lsak, o'zgaradigan ob'ektlarni ko'rsatkich yoki pointer sifatida kiruvchi argument qilib funksiyaga berishimiz mumkin. Bundan tashqari biz funksiyadan &-ko'rsatkichni qaytarishimiz mumkin. Lekin bu yersa ehtiyot bo'lish zarur, ko'rsatkich funksiya ichidagi static o'zgaruvchiga ko'rsatib turishi lozim. Chunki oddiy o'zgaruvchilar avtomatik ravishda funksiya tugaganida hotiradan olib tashlanadi. Buni misolda ko'raylik.

```
...
int& square(int k){
  static int s = 0; //s static sifatiga ega bo'lishi shart;
  int& rs = s;
    s = k * k;
  return (rs);
}
 ...
int g = 4;
int j = square(g); // j = 16
...
```

FUNKSIYA ARGUMENTLARNING BERILGAN QIYMATLARI

Ba'zi bir funksiyalar ko'pincha bir hil qiymatli argumentlar bilan chaqirilishi mumkin. Bu holda, agar biz funksiya argumentlariga ushbu ko'p qo'llaniladigan qiymatlarni bersak, funksiya argumentsiz chaqirilganda bu qiymatlar kompilyator tomonidan chaqiriqqa kiritiladi. Berilgan qiymatlar funksiya prototipida berilsa kifoyadir. Berilgan qiymatli argumentlar parametrlar ro'hatida eng o'ng tomonda yozilishi kerak. Buning sababi shuki, agar argument qiymati tashlanib o'tilgan bo'lsa, va u o'ng tomonda joylashmagan bo'lsa, biz bo'sh vergullani qo'yishimizga to'g'ri keladi, bu esa mumkin emas. Agar bir necha berilgan qiymatli argumentlar bor bo'lsa, va eng o'ngda joylashmagan argument tushurilib qoldirilsa, undan keyingi argumentlar ham yozilmasligi kerak. Bir misol keltiraylik.

```
 //Berilgan qiymatli parametrlar bilan ishlash
# include <iostream.h>
 int square(int = 1, int = 1); // ...(int a=1, int b=1)...
                          // yuqoridagi kabi o'zgaruvchilar otini ham
                          // berishimiz mumkin
 int main()
{
  int s = 3, t = 7;
  cout << "Paremetrsiz: " << square()<< endl;
  cout << "Bitta parametr (ikkinchisi) bilan:" << square(t) << endl;
  cout << "Ikkita parametr bilan:" << square(s,t) << endl;
  return (0);
}
 int square(int k, int g){
  return ( k * g );
}
```

Ekranda:

```
 Parametrsiz: 1
Bitta parametr (ikkinchisi) bilan: 7
Ikkita parametr bilan: 21
```

FUNKSIYA ISMI YUKLANISHI

Bir hil ismli bir necha funksiya e'lon qilinishi mumkin. Bu C++ dagi juda kuchli tushunchalardandir. Yuklatilgan funksiyalarning faqat kirish parametrlari farqli bo'lishi yetarlidir. Qaytish parametri yuklatilishda ahamiyati yo'qdir. Yuklangan funksiyalar chaqirilganda, qaysi funksiyani chaqirish kirish parametrlarining soniga, ularning tipiga va navbatiga bog'liqdir. Yani ism yuklanishida funksiyaning imzosi rol o'ynidi. Agar kirish parametrlari va ismlari ayni funksiyalarning farqi faqat ularning qaytish qiymatlarida bo'lsa, bu yuklanish bo'lmaydi, kompilyator buni hato deb e'lon qiladi. Funksiya yuklanishi asosan ayni ishni yoki amalni farqli usul bilan farqli ma'lumot tiplari ustida bajarish uchun qo'llaniladi. Masalan bir fazoviy jismning hajmini hisoblash kerak bo'lsin. Har bir jismning hajmi farqli formula yordamida, yani farqli usulda topiladi, bir jismda radius tushunchasi bor bo'lsa, boshqasida asos yoki tomon tushunchasi bor bo'ladi, bu esa farqli ma'lumot tiplariga kiradi. Lekin amal ayni - hajmni hisoblash. Demak, biz funksiya yuklanishi mehanizmini qo'llasak bo'ladi. Bir hil amalni bajaruvchi funksiyalarni ayni nom bilan atashimiz esa, dasturni o'qib tushunishni osonlashtiradi. Kompilaytor biz bergan funksiya imzosidan (imzoga funksiya ismi va kirish parametrlari kiradi, funksiyaning qaytish qiymati esa imzoga kirmaydi)

yagona ism tuzadi, dastur ijrosi davruda esa funksiya chaqirig'idagi argumentlarga qarab, kerakli funksiyani chaqiradi. Yangi ismni tuzish operatsiyasi ismlar dekoratsiyasi deb ataladi. Bu tushunchalarni misolda ko'rib chiqaylik.

```
// Yuklatilgan funksiyalarni qo'llash
 # include <iostream.h>
# include <math.h>
    // Yangi ismlar sohasini aniqladik
namespace
  mathematics {
     const double Pi = 3.14159265358979;
 double hajm(double radius); // sharning hajmi uchun - 4/3 * Pi * r^3
double hajm(double a, double b, double s) // kubning hajmi uchun - abc
 }
 using namespace mathematics;
 int main()
{
  double d = 5.99; // sharning radiusi
  int x = 7, y = 18, z = 43;
   cout << "Sharninig hajmi: " << hajm(d) << endl;
  cout << "Kubning hajmi: " << hajm(x,y,z) << endl;
  return (0);
}
 double mathematics::hajm(double radius) {
  return (  (Pi * pow(radius,3) * 4.0) / 3.0  );
}
 double mathematics::hajm(double a, double b, double c) {
  return ( a * b * c );
}
```

Ekranda:

```
 Sharning hajmi: 900.2623
Kubning hajmi: 5418
```

Yuqoridagi dasturda yangi ismlar sohasini aniqladik, unda Pi konstantasini e'lon qildik. shaqning hajmini hisoblashda standart kutubhonadagi pow() funksiyasini ishlatdik, shu sababli <math.h> e'lon faylini # include ifodasi bilan kiritdik. Ismlar sohasida joylashgan funksiyalarni aniqlash uchun, yani ularning tanasini yozish uchun biz ilarning to'liq ismini berishimiz kerak. Albatta, agar funksiya ismlar sohasining ichida aniqlangan bo'lsa, tashqarida boshqattan yozib o'tirishning hojati yo'q. hajm() funksiyalarining to'liq ismi mathematics::hajm(...) dir. :: operatori sohalarni bog'lovchi operatordir. Yangi ismlar sohasini faqatgina misol tariqasida berdik, uni funksiya yuklanishlari bilan hech qanday aloqasi yo'qdir. Funksiya ismlari yuklanishi, ko'rib turganimizdek, juda qulay narsadir. Funksiya yuklanishini qo'llaganimizda, funksiyalar argumentlarining berilgan qiymatlarini ehtiyotkorlik bilan qo'llashimiz lozim. Masalan bizda ikkita funksiyamiz bor bo'lsin.

```
foo(int k = 0); // berilgan qiymati 0
foo();
```

Bu ikki funksiya yuklatilgan. Lekin agar biz birinchi funksiyani dasturda argumentsiz chaqirsak, kompilyator qaysi funksiyani chaqirishni bilmaydi, va shu sababli hato beradi. Biroq bu deganimiz funksiya yuklanishi bilan berilgan qiymatlar qo'llanilishi mumkin emas deganimiz emas, eng muhimi funksiya chaqirig'ini ikki hil tushunish bo'lmasligi kerak.

FUNKSIYA SHABLONLARI

Funksiya shablonlari (function templates) ham funksiya yuklanishiga o'hshash tushunchadir. Bunda eng asosiy farq funksiya shablonlarida amal ham bir hil yo'l bilan bajariladi. Masalan bir necha sonlar ichidan eng kattasini topish kerak bo'lsin. Sonlar to'plami faqat tipi bilan farqlanadi, int, double yoki float. Ishlash algoritmi esa aynidir. Bu holda biz funksiyalarni yuklab o'tirmasdan, shablon yozib qo'ya qolamiz. Funkisya shabloni yoki yuklanishisiz ham bu masalani yechish mumkinku degan savol paydo bo'ladi. Masalan, agar biz kiradigan parametrlarning hammasini long double qilsak, istalgan sonli tipdagi argumentni bera olamiz, chunki kompilyator o'zi avtomatik ravishda kirish tiplarini *long double* ga o'zgartiradi. Lekin, agar biz bunday funksiya yozadigan bo'lsak, hotiradan va tezlikdan yutqizamiz. Dasturimizda faqat char tipidagi, bir baytli qiymatlar bilan ishlashimiz mumkin. long double esa 10 bayt, va eng katta sonni aniqlash uchun sonlarni solishtirganimizda, long double qiymatlarni solishtirish char tipidagi qiymatlarni solishtirishdan ko'ra ancha ko'p vaqt oladi. Qolaversa, har doim ham kompilyator tiplarni biridan ikkinchasiga to'g'ri keltira oladi. Shablonlarning strukturasi bilan tanishaylik. Bizning funksiya ikkita kirish argumentini bir biriga qo'shsin, va javobni qaytarsin.

```
template <class T>
T summa(T a, T b) {
  return ( a + b);
}
```

Shablon funksiya e'loni va aniqlanishidan oldin template <> ifodasi yoziladi, <> qavslardan keyin nuqta-vergul (;) qo'yilmaydi. <> qavslar ichida funksiya kirish parametrlari, chiqish qiymati va lokal o'zgaruvchilar tiplari beriladi. Ushbu formal tiplarning har birining oldida class yoki typename (tip ismi) so'zi qo'yilish kerak. Yuqoridagi misolda T ning o'rniga istalgan boshqa identefikator qo'yish mumkin. Misollar beraylik.

```
template <class javob, class uzunlik, class englik, class balandlik>
javob hajmKub(uzunlik a, englik b, balandlik c);
 template <typename T>
T maximum(T k, T l);
```

Yuqorida yozgan shablonimizni qo'llagan holga bir misol keltiraylik.

```
// Shablonlar bilan ishlash
# include <iostream.h>
template <class T>
T summa(T a, T b) {
  return ( a + b );
}
int main()
{
int x = 22, y = 456;
float m = .01, n = 56.90; // kasrli sonda nuqtadan oldingi (butun qismdagi)
                    // nolni berish shart emas: ... m = .01 ...
  cout << "int: 22 + 456 = " << summa(x,y) << endl;
  cout << "float: 0.01 + 56.90 = " << summa(0.01,56.90) << endl;
  return (0);
}
```

Ekranda:

```
int: 22 + 456 = 478
float: 0.01 + 56.90 = 56.91
```

Shablonlarni funksiyalardan tashqari klaslarga ham qo'llasa bo'ladi. Ko'rib turganimizdek, shablonlar faqat bir marotaba yoziladi. Keyin esa mos keladigan tiplar qo'yilib, yozilib ketilaveradi. Aslida shablonlar C++ ning standartida juda ko'p qo'llanilgan. Agar bilib ishlatilsa, shablonlar dasturchining eng kuchli quroliga aylanishi mumkin. Biz keyinroq yana shablonlar mavzusiga qaytamiz.

MASSIVLAR

Bu qismda dasturdagi ma'lumot strukturalari bilan tanishishni boshlaymiz. Dasturda ikki asosiy tur ma'lumot strukturalari mavjuddir. Birinchisi statik, ikkinchisi dinamikdir. Statik deganimizda hotirada egallagan joyi o'zgarmas, dastur boshida beriladigan strukturalarni nazarda tutamiz. Dinamik ma'lumot tiplari dastur davomida o'z hajmini, egallagan hotirasini o'zgartirishi mumkin. Agar struktura bir hil kattalikdagi tiplardan tuzilgan bo'lsa, uning nomi massiv (array) deyiladi. Massivlar dasturlashda eng ko'p qo'laniladigan ma'lumot tiplaridir. Bundan tashqari strukturalar bir necha farqli tipdagi o'zgaruvchilardan tashkil topgan bo'lishi mumkin. Buni biz klas (Pascalda record) deymiz. Masalan bunday strukturamiz ichida odam ismi va yoshi bo'lishi mumkin. Bu bo'limda biz massivlar bilan yaqindan tanishib o'tamiz. Bu bo'limdagi massivlarimiz C uslubidagi, pointerlarga (ko'rsatkichlarga) asoslan strukturalardir. Massivlarning boshqa ko'rinishlarini keyingi qismlarda o'tamiz. Massivlar hotirada ketma-ket joylashgan, bir tipdagi o'zgaruvchilar guruhidir. Alohida bir o'zgaruvchini ko'rsatish uchun massiv nomi va kerakli o'zgaruvchi indeksini yozamiz. C/C++ dagi massivlardagi elementlar indeksi har doim noldan boshlanadi. Bizda char tipidagi m nomli massiv bor bo'lsin. Va uning 4 dona elementi mavjud bo'lsin. Shemada bunday ko'rsataylik:

```
m[0] ->  4
m[1] -> -44
m[2] -> 109
m[3] ->  23
```

Ko'rib turganimizdek, elementga murojaat qilish uchun massiv nomi va [] qavslar ichida element indeksi yoziladi. Bu yerda birinchi element qiymati 4, ikkinchi element - 1 nomerli indeksda -44 qiymatlari bor ekan. Ohirgi element indeksi n-1 bo'ladi (n - massiv elementlari soni). [] qavslar ichidagi indeks butun son yoki butun songa olib keluvchi ifoda bo'lmog'i lozim. Masalan:

```
int k = 4, l = 2;
m[ k-l ] = 77; // m[2] = 77
m[3] *= 2; // m[3] = 46
double d = m[0] * 6; // d = 24
cout << m[1]; // Ekranda: -44
```

Massivlarni ishlatish uchun ularni e'lon qilish va kerak bo'lsa massiv elementlarini initsalizatsiya qilish kerak. Massiv e'lon qilinganda kompilyator elementlar soniga teng hajmda hotira ajratadi. Masalan yuqorida qo'llanilgan char tipidagi m massivini e'lon qilaylik: *char m[4];* Bu yerdagi 4 soni massivdagi elementlar miqdorini bildiradi. Bir necha massivni e'londa bersak ham bo'ladi:

```
int m1[4], m2[99], k, l = 0;
```

Massiv elementlari dastur davomida initsalizatsiya qilishimiz mumkin, yoki boshlang'ich qiymatlarni e'lon vaqtida, {} qavslar ichida ham bersak bo'ladi. {} qavslardagagi qiymatlar massiv initsalizaytsiya ro'yhati deyiladi.

```
int n[5] = {3, 5, -33, 5, 90};
```

Yuqorida birinchi elementning qiymati 3, ikkinchiniki 5 ... ohirgi beshinchi element qiymati esa 90 bo'ldi. Boshqa misol: ***double array[10] = {0.0, 0.4, 3.55};*** Bu yerdagi massiv tipi double bo'ldi. Ushbu massiv 10 ta elementdan iboratdir. {} qavslar ichida esa faqat boshlangich uchta element qiymatlari berildi. Bunday holda, qolgan elementlar avtomatik tarzda nolga tenglashtiriladi. Bu yerda aytib o'tishimiz kerakki, {} qavslar ichida berilgan boshlangish qiymatlar soni massivdagi elementlar sonidan katta bo'lsa, sintaksis hatosi vujudga keladi. Masalan:

```
char k[3] = {3, 4, 6, -66, 34, 90}; // Hato!
```

Uch elementdan iborat massivga 6 dona boshlangich qiymat berilyapti, bu hatodir. Boshqa misolni ko'rib chiqaylik:

```
int w[] = {3, 7, 90, 78};
```

w nomli massiv e'lon qilindi, lekin [] qavslar ichida massivdagi elementlar soni berilmadi. Bunday holda necha elementga joy ajratishni kompilyator {} qavslar ichidagi boshlangich qiymatlar miqdoriga qarab biladi. Demak, yuqoridagi misolda w

massivimiz 4 dona elementdan iborat bo'ladi. E'lon davridagi massiv initsalizatsiya ro'yhati dastur ijrosi vaqtidagi initsalizatsiyadan ko'ra tezroq ishlaydigan mashina kodini vujudga keltiradi.
Bir misol keltiraylik.

```
// Massivlar bilan ishlash.
# include <iostream.h>
# include <iomanip.h>
const int massiv = 8; // massiv kattaligi uchun konstanta
int k[massiv];
char c[massiv] = {5,7,8,9,3,44,-33,0}; // massiv initsalizatsiya ro'yhati
int main()
{
  for (int i = 0; i < massiv; i++) {
    k[i] = i + 1;              // dastur ichida inisalizatsiya
  }
   for (int j = 0; j < massiv; j++) {
    cout << k[j]
        << setw(4)
        << c[j]
        << endl;
  }
   return (0);
}
```

Ekranda:

```
1   5
2   7
3   8
4   9
5   3
6  44
7 -33
8   0
```

Yuqorida <iomanip.h> faylini dasturimizga kiritdik. Bu e'lon faylida standart kirish/chiqish oqimlari bilan ishlaydigan buyruqlar berilgan. Dasturimizda qo'llanilgan setw() manipulyatori chiqish oqimiga berilayatgan ma'lumotlarning eng kichik kengligini belgilaydi, biz setw() parametrini 4 deb berdik, demak c[] massivi elementlari 4 harf kenglikda ekranga bosiladilar. Agar kenglik kamlik qilsa, u kattalashtiriladi, agar bo'sh joy qolsa, elementlar chapga yondashilib yoziladi. Biz <iomanip.h> va manipulyatorlarni keyinroq to'la ko'rib chiqamiz. Misolimizda massiv nomli konstantani qo'lladik. Uning yordamida massiv chegaralarini va for strukturasidagi chegaraviy qiymatlarni berdik. Bunday o'zgarmasni qo'llash dasturda hatoga yo'l qo'yishni kamaytiradi. Massiv chegarasi o'zgarganda, dasturning faqat bir joyiga o'zgarish kiritiladi. Massiv hajmi e'lonida faqat const sifatli o'zgaruvchilar qo'llanilishi mumkin. Massivlar bilan ishlaganda eng ko'p yo'l qoyiladigan hato bu massivga 0 dan kichkina va (n-1) dan (n: massivdagi elementlar soni) katta indeks bilan murojaat qilishdir. Bunday hato dastur mantig'i hatosiga olib keladi. Kompilyator bu turdagi hatolarni tekshirmaydi. Keyinroq o'zimiza yozgan massiv klaslarida ushbu hatoni tekshiriladigan qilishimiz mumkin. 10 ta sonning tushish ehtimilini ko'rsaturvchi dastur yozaylik.

```
// Ehtimollar va massivlar
# include <iomanip.h>
# include <iostream.h>
# include <time.h>
# include <stdlib.h>
int main ()
{
  const int massivHajmi = 10;
  int m[massivHajmi] = {0}; // hamma 10 ta element
                    // 0 ga tenglashtirildi
  srand( time(NULL) );
  for(int i = 0; i < 1000; i++) {
    ++m[ rand() % 10 ];
  }
  for(int j = 0; j < massivHajmi; j++) {
    cout << j << setw(4) << m[j] << endl;
  }
  return (0);
}
```

Ekranda:

```
0  96
1  89
2 111
```

```
3  97
4 107
5  91
6 100
7 118
8  99
9  92
```

Ko'rib turganimizdek, sonlarning tushish ehtimoli nisbatan tengdir. Albatta, bu qiymatlar dasturning har yangi ishlashida o'zgaradi.

```
 ++m[ rand() % 10 ];
```

Yozuvi bilan biz massivning rand() % 10 indeksli elementini birga oshirmoqdamiz. Bunda rand () % 10 ifodasidan chiqadigan qiymatlar [0;9] ichida yotadi.

Satrlar, yani harflar ketma-ketligi ("Toshkent", "Yangi yilingiz bilan!"...) C/C++ da char tipidagi massivlar yordamida beriladi. Bunday satrlar bilan islovlar juda tez bajariladi. Chunki ortiqcha tekshirishlar bajarilmaydi. Bundan tashqari C++ da ancha rivojlangan String klasi mavjuddir, u oddiy char bilan berilgan satrlardan ko'ra qulayroqdir. Lekin ushbu klas ko'proq joy egallaydi va massivli satrlardan ko'ra sekinroq ishlaydi. String klasini keyingi qismlarda o'tamiz. Qolaversa, satrlar bilan ishlash uchun biz o'zimiz klas yoki struktura yozishimiz mumkin. C dan meros bo'lib qolgan satrlar ustida amallar bajarish uchun biz dasturimizga <string.h> (yangi ismi <cstring>) e'lon faylini kiritishimiz kerak. Ushbu e'lon faylida berilgan funksiyalar bilan keyingi bo'limda ishlaymiz. Harflar, yani literalar, aytib o'tganimizdek, C++ da char tipi orqali beriladi. Literalar apostroflarga ('S', '*' ...) olinadi. Satrlar esa qo'shtirnoqlarga olinadi. Satrlar e'loniga misol beraylik.

```
 char string[] = "Malibu";
char *p = "Ikkinchi!?!";
```

Satrlarni yuqoridagi ikkita yo'l bilan initsalizatsiya qilsa bo'ladi. Satrlar ikkinchi uslubda e'lon qilinganda, yani pointer mehanizmi qo'llanilganda, ba'zi bir kompilyatorlar satrlarni hotiraning konstantalar saqlanadigan qismiga qo'yadi. Yani ushbu satrlarga o'zgartirish kiritilishi ta'qiqlanadi. Shu sababli satrlarni hardoim o'zgartirish imkoni bo'lishi uchun ularni char tipidagi massivlar ko'rinishida e'lon qilish afzaldir. Satrlar massiv yordamida berilganda, ularning uzunligi noma'lumdir. Shu sababli satrning tugaganligini bildirish uchun satr ohiriga mahsus belgi nol literasi qo'yiladi. Uning dastursa belgilanishi '\0' ko'rinishga ega. Demak, yuqorida berilgan "Malibu" satriga ohiriga '\0' belgisi qo'shiladi, yani massivning umumiy uzunligi "Malibu":6 + '\0':1 = 7 ta char tipidagi elementga teng bo'ladi. Satrlarni massiv initsalizatsiya ro'yhati ko'rinishida ham bersak bo'ladi:

```
 char c[6] = {'A', 'B', 'C', 'D', 'E' , '\0'};
...
cout << c;
...
```

Ekranda:

```
 ABCDE
```

Biz cout bilan c ning qiymati ekranga bosib chiqardik. Aynan shunday klaviaturadan ham o'qib olishimiz mumkin:

```
char string[80];
cin >> string;
```

Eng muhimi satr bilan '\0' belgisi uchun yetarli joy bo'lishi kerak. Satrlar massiv yordamida berilganligi uchun, alohida elementlarga indeks orqali yetishish mumkin, masalan:

```
 char k[] = "Bahor keldi, gullar ochildi.";
for (int i = 0; k[i] != '\0'; i++)
   if ( (i mod 2) == 0 )     // juft sonlar uchun haqiqat bo'ladi
      cout << k[i] << " ";
```

Ekranda:

```
 B h r k l i   u l r o h l i
```

Yuqoridagi misolda, sikl tugashi uchun k[i] element '\0' belgiga teng bo'lishi kerak.

FUNKSIYALARNING MASSIV KIRISH PARAMETRLARI

Funksiyalarga massivlarni kirish argument sifatida berish uchun parametr e'lonida [] qavslar qo'yiladi. Masalan:

```
...
void sortArray(int [], int );  // funksiya e'loni
void sortArray(int n[], int hajm) { // funksiya aniqlanishi
 ...
}
...
```

Dasturda esa, funksiya chaqirilganda, massivning faqat ishmi beriladi halos, [] qavslarning keragi yo'q.

```
 int size = 10;
int array[size] = {0};
 ...
void sortArray(array, size); // funksiya chaqirig'i,
```

```
                // faqat massiv ismi - array berildi
...
```

Funksiyaga massivlarni berganimizda, eng katta muammo bu qanday qilib massivdagi elementlari sonini berishdir. Eng yaxshi usul bu massiv kattaligini qo'shimcha kirish parametri orqali funksiyaga bildirishdir. Bundan tashqari, massiv hajmini global konstanta orqali e'lon qilishimiz mumkin. Lekin bu ma'lumotni ochib tashlaydi, global sohani ortiqcha narsalar bilan to'ldirib tashlaydi. Undan tashqari massiv hajmini funksiyaning o'ziga yozib qoyishimiz mumkin. Biroq bunda bizning funksiyamiz faqat bitta kattalikdagi massivlar bilan ishlaydigan bo'lib qoladi. Yani dasturimiz dimamizmni yo'qotadi. Klaslar yordamida tuzilgan massivlar o'z hajmini biladi. Agar bunday ob'ektlarni qo'llasak, boshqa qo'shimcha parametrlarni qo'llashimizning keragi yo'q. Funksiyalarga massivlar ko'rsatkich ko'rinishida beriladi. Buni C++, biz ko'rsatmagan bo'lsak ham, avtomatik ravishda bajaradi. Agar massivlar qiymat bo'yicha chaqirilganda edi, har bir massiv elementining nushasi olinishi kerak bo'lardi, bu esa dastur ishlash tezligiga salbiy ta'sir ko'rsatar edi. Lekin massivning alohida elementi argument o'rnida funksiyaga berilganda, ushbu element, aksi ko'rsatilmagan bo'lsa, qiymat bo'yicha beriladi. Masalan:

```
 ...
double m[3] = {3.0, 6.88, 4.7};
void foo(double d){
   ...
}
...
int main()
{
...
void foo(m[2]); // m massivining uchinchi elementining qiymati - 4.7 berildi
...
return (0);
}
```

Agar kiritilayatgan massiv funksiya ichida o'zgarishi ta'qiqlansa, biz funksiya massiv parametri oldiga const sifatini qo'ysak bo'ladi:

```
 foo(const char []);
```

Bunda funksiyaga kiradigan massiv funksiya tomonidan o'zgartirilmaydi. Agar o'zgartirishga urinishlar bo'lsa, kompilyator hato beradi. Massivlar va funksiyalarning birga ko'llanilishiga misol beraylik.

```
// Massiv argumentli funksiyalar
 # include <istream.h>
 const int arraySize = 10;
 double ortalama(int m[], int size) {
   double temp = 0;
    for (int i = 0; i < size; i++) {
      temp += m[i];
   }
   return ( temp / size );
}
 void  printArray(const int n[], int size, int ortalama) {
   for (int i = 0; i < size; i++) {
      cout << n[i]; << endl;
   }
   cout << "O'rtalama: " << ortalama << endl;
}
 int main()
{
   int m[10] = {89,55,99,356,89,335,78743,44,767,346};
   printArray(m, arraySize, ortalama(m, arraySize)) ;
   return (0);
}
```

Ekranda:

```
 89
55
99
356
89
335
78743
44
767
346
O'rtalama: 8092.3
```

BIR NECHA INDEKSLI MASSIVLAR

Massivlar bir necha indeksga ega bo'lishlari mumkin. C++ kompilyatorlari eng kamida 12 ta indeks bilan ishlashlari mumkin. Masalan, matematikadagi m x n kattalikdagi matritsani ikkita indeksli massiv yordamida berisak bo'ladi.

```
int matritsa [4][10];
```

Yuqorida to'rt satrlik, 10 ustunlik matritsani e'lon qildik. Bir indeksli massivlar kabi ko'p indeksli massivlarni initsalizatsiya ro'yhati bilan birga e'lon qilish mumkin. Masalan:

```
char c[3][4] = {
                { 2,  3,9, 5}, // birinchi satr qiymatlari
                {-10, 77,5, 1}, // ikkinchi  "       "
                { 90,233,3,-3}  // uchinchi  "       "
              };
int m[2][2] = {56,77,8,-3}; // oldin birinchi satrga qiymatlar beriladi,
                            // keyin esa ikkinchi satrga
double d[4][3][6] = {2.55, -46,0988}; // birinchi satrning dastlabki ikkita
                                      // elementi qiymat oladi,
                                      // massivning qolgan elementlari esa
                                      // nolga tenglashtiriladi
```

Massivning har bir indeksi alohida [] qavslar ichiga olinishi kerak. Yuqoridagi c[][] massivining ikkinchi satr, birinchi ustunidagi elementi qiymatini birga oshirish uchun

```
++c[1][0]; // yoki c[1][0]++;
        //      c[1][0] += 1;
        //      c[1][0] = c[1][0] + 1;
```

deb yozishimiz mumkin. Massiv indekslari 0 dan boshlanishini unutmaslik zarur.

Agar

```
++c[1,0];
```

deb yozganimizda hato bo'lar edi. C++ bu yozuvni

```
++c[0];
```

deb tushunar edi, chunki kompilyator vergul bilan ajratilgan ro'yhatning eng ohirgi elementini qabul qilardi. Hullas, C++ dagi ko'p indeksli massivlar dasturchiga behisob imkoniyatlar beradi. Undan tashqari, ular hotirada statik joylashganligi uchun ularning ishlash tezligi kattadir. C++ dagi ko'p indeksli massivlar hotirada ketma-ket joylashgandir. Shu sababli agar massiv funksiyaga kirish parametri sifatida berilsa, faqat birinchi indeks tushurilib qoldiriladi, qolgan indekslar esa yozilishi shartdir. Aks taqdirda funksiya massiv kattaligini to'g'ri keltirib chiqarolmaydi. Massiv parametrli bir funksiya e'lonini beraylik.

```
//Ko'p indeksli massivlar
 # include <iostream.h>
 int indeks = 3;
int intArray[indeks][4] = {}; // hamma elementlar 0 ga tenglashtirildi
 void printArray(int mass[][4], int idx){   // funksiya e'loni
    for (int i = 0; i < idx; i++) { // massivning birinchi indeksini
                                    // o'zgartirsa bo'ladi
      for (int k = 0; k < 4; k++){ // massivning ikkinchi indeksi o'zgarmaydi
        cout << mass[i][k];
      }
      cout << endl;
   }
   return;
}
...
 int main()
{
...
printArray(intArray); // funksiya chaqirig'i
...
return (0);
}
```

Massivning indekslarini funksiyaga bildirish yana muammoligicha qoladi. Albatta, birinchi indeksdan tashqari qolgan boshqa indekslar kattaligini funksiya ichida berish ma'noga egadir. Lekin birinchi indeks kattaligini tashqaridan, qo'shimcha parametr sifatida bersak, funksiyamiz chiroyliroq chiqadi, turli kattalikdagi massivlarni o'lish imkoniga ega bo'ladi.

POINTER (ko'rsatkich) VA SATRLAR

C++ da ikki ko'rinishdagi ko'rsatkichlar - &-ko'rsatkichlar va pointerlar mavjuddir. Aslida bularning farqi faqat qo'llanilishi va ko'rinishida desak ham bo'ladi. Bu qismda biz C dan meros qolgan pointerlar bilan yaqindan tanishamiz. &-ko'rsatkichlarni biz o'tgan qismda ko'rgan edik. Pointerlar C/C++ dasturlash tillarining eng kuchli qurollaridandir. Lekin pointer tushunchasini anglash ham oson emas. Pointerlar yordamida funksiyalarning ko'rsatkich bo'yicha chaqirish mehanizmini amalga oshirish mumkin. Undan tashqari pointerlar yordamida dinamik strukturalar – stek (stack), ro'yhat (list), navbat (queue) va darahtlar (tree) tuzish mumkin. Undan tashqari pointer, satr va massivlar orasida yaqin aloqa bordir. Satr va massivlarni pointerlar yordamida berish bizga C dan meros bo'lob qoldi. Keyingi boblarda biz satr va massivlarni to'la qonli ob'ekt sifatida qo'lga olamiz. Pointerlar qiymat sifatida hotira adreslarini oladilar. Oddiy o'zgaruvchilar ma'lum bir qiymatga

ega bo'lgan bir paytda, pointerlar boshqa bir o'zgaruvchining adresini o'z ichlariga oladilar. Shunda o'zgaruvchi bevosita qiymatga ko'rsatib tursa, pointer qiymatga bilvosita ko'rsatadi. Pointer e'lonida, pointer ismidan oldin '*' (yulduzcha, ko'paytiruv) belgisi qo'yilishi kerak. Misolda ko'raylik:

```
char *charPtr, c = 8, *pc, ff[] = "IcyCool";
```

Bu yerda charPtr va pc lar char tipidagi ko'rsatkichlardir. Yani, charPtr ni "char tipidagi oz'garuvchiga ga ko'rsatkich" deb o'qisak bo'ladi. Ko'rsatkich sifatida e'lon qilinayatgan har bir o'zgaruvchi ismi oldida '*' bo'lishi shartdir. Pointerlarni boshqa o'zgaruvchilar kabi e'lon davrida, yoki dastur ichida qiymat berish yordamida initsalizatsiya qilish mumkin. Pointerlar qiymat sifatida faqat o'zgaruvchi yoki ob'ektlarning adreslarini va NULL yoki 0 ni oladilar. NULL simvolik konstantasi <iostream.h> va boshqa bir necha standart e'lon fayllarida aniqlangan. Pointerga NULL qiymatini berish pointerni 0 ga tenglashtirish bilan tengdir, ammo C++ pointerga to'g'ridan-to'g'ri 0 qiymatini berish afzalroqdir, chunki agar NULL emas, 0 qiymati berilsa, ushbu qiymat avtomatik ravishda pointerning tipiga keltiriladi. Butun sonlardan faqat 0 qiymati pointerga keltirilishsiz berilishi mumkin. Agar ma'lum bir hotira adresini pointerga bermoqchi bo'lsak, quyidagicha yozishimiz mumkin:

```
int *iPtr, address = 0x45ad7 ;
iPtr = (int *) address;             // C   uslubida tiplarni keltirish
iPtr = static_cast<int *>(address); // C++    "       "         "
```

Lekin, albatta, yuqoridagi yozganimizni kamdam-kam qo'llashga to'g'ri kelsa kerak, chunki adres olishning soddaroq yo'llari bordir. Aslida, pointer e'lon qilinsa-yu lekin hali qo'llanilmayatgan bo'lsa, unga 0 qiymatini berish tavsiya etiladi. Chunki, agar biz bu 0 qiymatli pointerni bilmasdan qo'llamoqchi bo'lsak, kompilyator hato beradi, bunga sabab odatda operatsiyon sistemalar 0 adresli hotira maydoni bilan ishlashga ruhsat bermaydilar. Shu tariqa qiymatsiz pointer qo'llanilganidan ogoh bo'lamiz.

POINTER OPERATORLARI

O'zgaruvchilarning (yani harqanday ob'ektning) adresini olishda biz & operatorini - adres olish operatorini qo'llaymiz. Bu kontekstda & operatori bir dona argument oladi. Undan tashqari & ikkili operatori bitli qo'shishda qo'llaniladi. Adres olishga misol keltiraylik.

```
int *iPtr, var = 44;
iPtr = &var;
 double d = 77.0, *dPtr = &d;
```

Bu yerda bir narsani o'tib ketishimiz kerak. C++ da identefikatorlar (o'zgaruvchi va ob'ektlar) ikki turda bo'ladi. Birinchisi chap identefikatorlar (lvalue - left value: chap qiymat), ikkinchi tur esa o'ng identefikatorlardir (rvalue - right value: o'ng qiymat). Yani chap identefikatorlar '=' (qiymat berish operatori) belgisining chap argumenti sifatida qo'llanilishi mumkin. O'ng identifikatorlar esa '=' ning o'ngida qo'yilishlari kerak. Bunda o'ng identefikatorlar har doim ham chap identefikator o'rnida qo'llanila olomaydilar. Yani chap identefikatorlarning qiymatlari '=' operatori yordamida o'zgartirilishi mumkin. Agar o'zgaruvchi *const* sifati bilan e'lon qilingan bo'lsa, u normal sharoitda faqat o'ng identefikatordir. Bu ma'lumotlarni keltirganimizning sababi, & adres olish operatori faqat chap identefikator bo'la oladigan o'zgaruvchilarga nisbatan qo'llanilishi mumkin. Agar o'zgaruvchi const sifatli konstantalarga, register sifatli o'zgaruvchilarga va ko'rsatkich qaytarmaydigan (adres qaytarmaydigan) ob'ektlarga nisbatan qo'llanilishi ta'qiqlanadi. Faraz qilaylik, biz pointerimizga boshqa bir o'zgaruvchining adresini berdik. Endi bizning pointerimiz ikki qiymatni ifoda etmoqda, biri bu o'zining qiymati, yani boshqa o'zgaruvchining adresi, ikkinchi qiymat esa, bu boshqa o'zgaruvchining asl qiymatidir. Agar biz pointerning o'zi bilan ishlasak, biz hotiradagi adres bilan ishlagan bo'lamiz. Ko'p hollarda esa buning keragi yo'q. Pointer ko'rsatayotgan o'zgaruvchining qiymati bilan ushbu pointer yordamida ishlash uchun biz '*' belgisini, boshqacha qilib etganda, ko'rsatish operatorini (indirection, dereferencing operator) qo'llashimiz kerak bo'ladi. Bunga misol beraylik.

```
 ...
int k = 74;
int *kPtr = &k;
...
cout << k << " --- " << *kPtr << endl; // Ekranda:
                                 // 74 --- 74
cin >> *kPtr;                    // 290 qiymatini kiritaylik...
cout << k << " === " << *kPtr << endl; // Ekranda:
                                 // 290 === 290
*kPtr = 555;
cout << k << " ... "<< *Ptr << endl;   // Ekranda:
                                 // 555 ... 555
...
```

Ko'rib turganimizdek, biz kPtr ko'rsatkichi orqali k o'zgaruvchining ham qiymatlarini o'zgartira oldik. Demak *kPtr chap identefikatordir, chunki uning qiymatini qiymat berish operatori yordamida o'zgartira olar ekanmiz. Agar pointerimiz 0 ga teng bo'lsa, uni ko'rsatish operatori - '*' bilan ko'llash ko'p hollarda dastur ijrosi hatolariga olib keladi. Undan tashqari, boshlangich qiymatlari aniqlanmagan pointerni qiymatiga ko'rsatish eng kamida mantiqiy hatolarga olib keladi, bunung sababi, pointer ko'rsatayotgan hotira qismida oldingi ishlagan dasturlar kodlari qolgan bo'lishi mumkin. Bu esa bizga hech keragi yo'q. Pointer bo'lmagan o'zgaruvchilarga ko'rsatish operatorini qo'llash ta'qiqlanadi. Aslini olganda, & adres olish operatori va * ko'rsatish operatorlari, bir-birining teskarisidir. Bir misol kertiraylik.

```
 // Adres olish va ko'rsatish operatorlari
 # include <iostream.h>
 char c = 44;     // char tipidagi o'zgaruvchi
char *pc = &c;   // char tipidagi pointer
```

```
 int main ()
{
   cout << "&*pc ning qiymati: " << &*pc << endl;
   cout << "*&pc ning qiymati: " << *&pc << endl;
   cout << "c ning hotiradagi adresi: " << &c << endl;
   cout << "pc pointerning qiymati: " << pc << endl;
   cout << "c ning qiymati: " << c << endl;
   cout << "*pc ning qiymati: " << *pc << endl;
    return (0);
}
```

Ekranda:

```
&*pc ning qiymati: 0x4573da55
*&pc ning qiymati: 0x4573da55
c ning hotiradagi adresi: 0x4573da55
pc pointerning qiymati: 0x4573da55
c ning qiymati: 44
*pc ning qiymati: 44
```

Demak, &*pc va *&pc ayni ishni bajarar ekan, yani * va & operatorlari bir-birining teskarisidir. Hotiradagi adres ekranga boshqa ko'rinishda chiqishi mumkin. Bu mashina va kompilyatorga bog'liqdir.

POINTER ARGUMENTLI FUNKSIYALAR

Funksiylar ikki argumentlariga qarab ikki turga bo'linadi degan edik. Argumentlar qiymat bo'yicha, yoki ko'rsatkich bo'yicha berilishi mumkin edi. Qiymat bo'yicha berilgan argumentning funksiya chaqirig'iga nushasi beriladi. Ko'rsatkich bo'yicha argument chaqirig'ida, funksiyaga kerakli argumentga ko'rsatkich beriladi. Ko'rsatkich bo'yicha chaqiriqni ikki usulda bajarish mumkin, birinchi usul &-ko'rsatkichlar orqali amalga oshiriladi. Ikkinchi usulda esa pointerlar qo'llaniladi. Pointerlar bilan chaqishning afzalligi (qiymat bo'yicha chaqiriq bilan solishtirganda) shundagi, agar ob'ektlar katta bo'lsa, ulardan nusha olishga vaqt ketqizilmaydi. Undan tashqari funksiya ob'ektning asl nushasi bilan ishlaydi, yani ob'ektni o'zgartura oladi. Funksiya faqat bitta ob'ektni yoki o'zgaruvchini return ifodasi yordamida qiytara olgani uchun, oddiy yol bilan, qiymat bo'yicha chaqiriqda funksiya faqat bitta o'zgaruvchining qiymatini o'zgartira oladi. Agar pointerlarni qo'llasak, bittadan ko'p ob'ektlarni o'zgartirishimiz mumkin, huddi &-ko'rsatkichli chaqiriqdagi kabi. Funksiya chaqirig'ida esa, biz o'zgaruvchilarning adresini qo'llashimiz kerak. Buni & adres olish operatori yordamida bajaramiz. Massivni berayatganda esa adresni olish kerak emas, chunki massivning ismining o'zi massiv birinchi elementiga pointerdir.

Pointerlarni qo'llab bir dastur yozaylik.

```
// Pointer argumentli funksiyalar
# include <iostream.h>
 int foo1(int k) {return (k * k);}
void foo2(int *iPtr) {*iPtr = (*iPtr) * (*iPtr);}
int main()
{
   int qiymat = 9;
   int javob = 0;
    javob = foo1(qiymat);  // javob = 81
   cout << "javob = " << javob << endl;
    foo2(&qiymat);         // qiymat = 81
   cout << "qiymat = " << qiymat << endl;
   return (0);
}
```

Ekranda:

```
 javob = 81
qiymat = 81
```

Yuqoridagi dasturimizda foo2() funksiya chaqirig'ida qiymat nomli o'zgaruvchimizning adresini oldik (& operatori) va funksiya berdik. foo2() funksiyamiz iPtr pointer argumentining qiymatini * operatori yordamidao'zgartiryapti. Funksiya e'lonida pointer tipidagi parametrlardan keyin o'zgaruvchi ismlarini berish shart emas. Masalan:

```
 int func(int * , char * ); // funksiya e'loni
int func(int *arg1, char *arg2); // funksiya e'loni
```

Yuqoridagi ikki e'lon aynidir. Aytib o'tkanimizdek, massivlarning ismlari birinchi elementlariga ko'rsatkichdir. Hatto, agar massiv bir indeksli bo'lsa, biz massivlar bilan ishlash uchun pointer sintaksisini qo'llashimiz mumkin. Kompilyator

```
 foo(int m[]);
```

e'lonini

```
 foo(int * const m);
```

e'loniga almashtiradi. Yuqoridagi m pointerini "int tipiga o'zgarmas pointer" deb o'qiymiz. const bilan pointerlarning qo'llanilishini alohida ko'rib chiqamiz.

const SIFATLI POINTERLAR

const ifodasi yordamida sifatlantirilgan o'zgaruvchining qiymatini normal sharoitda o'zgartira olmaymiz. const ni qo'llash dasturning hatolardan holi bo'lishiga yordam beradi. Aslida ko'p dasturchilar const ni qo'llashga o'rganishmagan. Shu sababli ular katta imkoniyatlarni boy beradilar. Bir qarashda const ning keragi yo'qdek tuyuladi. Chunki const ni qo'llash

dasturning hech qaysi bir yerida majburiy emas. Masalan konstantalarni belgilash uchun # define ifodasini qo'llasak bo'ladi, kiruvchi argumentlarni ham const sifatisiz e'lon qilsak, dastur mantig'i o'zgarishsiz qoladi. Lekin const kerak-kerakmas joyda o'zgaruvchi va ob'ektlarning holat-qiymatlarini o'zgartirilishidan himoyalaydi. Yani ob'ekt qiymatini faqat cheklangan funksiyalar va boshqa dastur bloklari o'zgartira oladilar. Bu kabi dasturlash uslubi esa, yani ma'lumotni berkitish va uni himoya qilish ob'ektli dasturlash falsafasiga kiradi. Ko'rsatkich qo'llanilgan funksiyalarda, agar argumentlar funksiya tanasida o'zgartirilmasa, kirish parametrlari const deb e'lon qilinishlari kerak. Masalan, bir massiv elementlarini ekranga bosib chiqaradigan funksiya massiv elementlarini o'zgartirishiga hojat yo'q. Shu sababli argumentdagi massiv const sifatiga ega bo'ladi. Endi, agar dasturchi adashib, funksiya tanasida ushbu massivni o'zgartiradigan kod yozsa, kompilyator hato beradi.
Yani bizning o'zgaruvchimiz himoyalangan bo'ladi. Bu mulohazalar boshqa tipdagi const sifatli funksiya kirish parametrlariga ham tegishlidir. Pointerlar bilan const ni to'rt hil turli kombinatsiya qo'llashimiz mumkin.

1. Oddiy pointer va oddiy o'zgaruvchi (pointer ko'rsatayatgan o'zgaruvchi).
2. const pointer va oddiy o'zgaruvchi.
3. Oddiy pointer va const o'zgaruvchi.
4. const pointer va const o'zgaruvchi.

Yuqoridagilarni tushuntirib beraylik. Birinchi kombinatsiyada o'zgaruvchini hech bir narsa himoya qilmayapti. Ikkinchi holda esa o'zgaruchining qiymatini o'zgartirsa bo'ladi, lekin pointer ko'rsatayotgan adresni o'zgartirish ta'qiqlanadi. Masalan massiv ismi ham const pointerdir. Va u ko'rsatayatgan massiv birinchi elementi-ni o'zgartirishimiz mumkin. Endi uchinchi holda pointerimiz oddiy, lekin u ko'rsatayatgan o'zgaruvchi himoyalan gandir. Va nihoyat, to'rtinchi variantda eng yuqori darajadagi o'zgaruvchi himoyasita'minlanadi. Yuqoridagi tushunchalarga misol berib o'taylik.

```
 // const ifodasi va pointerlar
 # include <iostream.h>
# include <ctype.h>
 int countDigits(const char *);  // oddiy pointer va const o'zgaruvchi
void changeToLowerCase(char *); // oddiy pointer va oddiy o'zgaruvchi
 int main()
{
   char m[] = "Sizni 2006 yil bilan tabriklaymiz!";
   char n[] = "TOSHKENT SHAHRI...";
    cout << m << endl <<"Yuqoridagi satrimizda " << countDigits(m)
        << " dona son bor." << endl << endl;
   cout << n << endl << "Hammasi kichik harfda:" << endl;
   changeToLowerCase(n);
   cout << n << endl;
   return (0);
}
 int countDigits(const char * cpc) { // satrdagi sonlar (0..9) miqdorini
                                // hisoblaydi
   int k = 0;
    for ( ; *cpc != '\0' ; cpc++){ // satrlarni elementma-element
                               // ko'rib chiqishning birinchi yo'li.
      if ( isdigit(*cpc) )  // <ctype.h> kutubhona funksiyasi
         k++;
   }
    return (k);
}
 void changeToLowerCase(char *pc) { // katta harflarni kichik harflarga
                                // almashtiruvchi funksiya

   while( *pc != '\0'){           // satrlarni elementma-element
                               // ko'rib chiqishning ikkinchi yo'li.
      *pc = tolower( *pc );  // <ctype.h> kutubhona funksiyasi
      ++pc; // pc keyingi harfga siljitildi
   }
   return;
}
```

Ekranda:

```
 Sizni 2006 yil bilan tabriklaymiz!
Yuqoridagi satrimizda 4 dona son bor.

TOSHKENT SHAHRI...
Hammasi kichik harfda:
toshkent shahri...
```

Yuqoridagi dasturda ikki funksiya aniqlangan. Change ToLowerCase()funksiyasining parametri juda oddiydir. Oddiy char tipidagi pointer. Ushbu pointer ko'rsatayotgan ma'lumot ham oddiydir. Ikkinchi funksiyamizda esa (countDigits()), pointerimiz oddiy, yani uning qiymati o'zgarishi mumkin, u hotiraning turli adreslariga ko'rsatishi mumkin, lekin u

ko'rsatayotgan o'zgaruvchi const deb e'lon qilindi. Yani pointerimiz ko'rsatayotgan ma'lumot ayni ushbu pointer yordamida o'zgartirilishi ta'qiqlanadi. Bizda yana ikki hol qoldi, ular quyida berilgan:
const pointer va const o'zgaruvchi
const pointer va oddiy o'zgaruvchi
Birinchi holga misol beraylik.

```
...
int m = 88, j =77;
const int * const pi = &m; // const pointer e'lon paytida
                    // initsalizatsiya qilinishi shartdir
...
m = 44;   // To'g'ri amal
*pi = 33; // Hato! O'zgaruvchi const deb belgilandi; birinchi const
pi = &j;  // Hato! Pointer const deb belgilandi; int * dan keyingi const
...
j = *pi;  // To'g'ri. const o'zgaruvchilarning
        // qiymatlari ishlatilinishi mumkin.
...
```

Yuqoridagi parchada const pointer va const o'zgaruvchili kombinatsiyani ko'rib chiqdik. Eng asosiysi, const pointer e'lon qilinganda initsalizatsiya bo'lishi shart. Bu qonun boshqa tipdagi const o'zgaruvchilarga ham tegishli. Ko'rib turganimizdek,

```
*pi = 33;
```

ifodasi bilan m ning qiymatini o'zgartirishga intilish bo'ldi. Lekin bu hatodir. Chunki biz pi pointeri ko'rsatib turgan hotira adresini o'zgarmas deb pi ning e'lonida birinchi const so'zi bilan belgilagan edik. Lekin biz o'zgaruvchining haqiqiy nomi bilan - m bilan o'zgaruvchi qiymatini o'zgartira olamiz. Albatta, agar m ham o'z navbatida const sifatiga ega bo'lmasa. Yani hotira adresidagi qiymatga ikkita yetishish yo'li mavjud bo'lsa, bular o'zgaruvchining asl nomi - m, va pi pointeri, bulardan biri orqali ushbu qiymatni o'zgartirsa bo'ladi, boshqasi o'rqali esa bu amal ta'qiqlanadi.
Keyin,
pi = &j;
ifoda bilan esa pi ga yangi adres bermoqchi bo'ldik. Lekin pointerimiz o'zgarmas bo'lgani uchun biz biz bu amalni bajara olmaymiz. Pointerlar va const ifodasining birga qo'llanilishining to'rtinchi holida const pointer va oddiy hotira adresi birga ishlatilinadi. Bunga bir misol:

```
int j = 84, d = 0;
int * const Ptr = &j; // e'lon davrida initsalizatsiya shartdir
*Ptr = 100;          // to'g'ri amal
Ptr = &d;           // Hato! Ptr ko'rsatayatgan
                // hotira adresi o'zgartilishi ta'qiqlanadi
```

Yuqorida Ptr ko'ratayatgan adresni o'zgartirsak, kompilyator hato beradi.Aslida, massiv ismlari ham ayni shu holga misol bo'la oladilar. Massiv ismi massivning birinchi elementiga const pointerdir. Lekin u ko'rsatayotgan massiv birinchi elementning qiymati o'zgartilishi mumkin.

POINTER VA ODDIY O'ZGARUVCHILARNING EGALLAGAN ADRES KATTALIGI

Pointerlar istalgan ichki asos tipga (char, int, double ...) yoki qollanuvchi belgilagan tipga (class, struct ...) ko'rsatishi mumkin. Albatta, bu turli tiplar hotirada turlicha yer egallaydilar. char bir bayt bo'lsa, double 8. Lekin bu tiplarga ko'rsatuvchi pointerlarning kattaligi bir hil 4 bayt. Bu kattalik turli adreslash turlariga qarab o'zgarishi mumkin, lekin bitta sistemada turli tipdagi ko'rsatkichlar bir hil kattalikga egadirlar. Buning sababi shuki, pointerlar faqat hotiraning adresini saqlaydilar. Hozirgi sistemalarda esa 32 bit bilan istalgan adresni aniqlash mumkin. Pointerlar oz'garuvchining faqat boshlangich baytiga ko'rsatadilar. Masalan, bizda double tipidagi o'zgaruvchi d bor bo'lsin. Va unga ko'rsatuchi pd pointerimiz ham e'lon qilingan bo'lsin. Pointerimiz d o'zgaruvchisining faqat birinchi baytiga ko'rsatadi. Lekin bizning d o'zgaruvchimizda pointerimiz ko'rsatayotgan birinchi baytidan tashqari yana 7 dona qo'shimcha bayti mavjud. Mana shunda pointerning tipi o'yinga kiradi. Kompilyator double tipi hotirada qancha joy egallishi bilgani uchun, pointer ko'rsatayotgan adresdan boshlab qancha baytni olishni biladi. Shuning uchun pointerlar hotirada bir hil joy egallasa ham, biz ularga tip beramiz. void * tipidagi pointerni ham e'lon qilish mumkin. Bu pointer tipsizdir. Kompilyator bu pointer bilan * ko'rsatish operatori qo'llanilganda necha bayt joy bilan ishlashni bilmaydi. Shu sababli bu amal tayqiqlangandir. Lekin biz void * pointerini boshqa tipdagi pointerga keltirishimiz mumkin, va o'sha yangi tip bilan ishlay olamiz.

Masalan:

```
...
int i = 1024;
int *pi = &i, *iPtr;
void *pv;
pv = (void *) pi;
cout << *pv; // Hato!
```

```
iPtr = (int *) pv;
cout << *iPtr; // Ekranda: 1024
...
```

Tiplarning hotiradagi kattaligini kopsatadigan, bir parametr oladigan sizeof() (sizeof - ning kattaligi) operatori mavjuddir. Uning yordamida tiplarning, o'zgaruvchilarning yoki massivlarning kattaliklarini aniqlash mumkin. Agar o'zgaruvchi nomi berilsa, () qavslar berilishi shart emas, tip, massiv va pointer nomlari esa () qavslar ichida beriladi. Bir misol beraylik.

```
// sizeof() operatori

# include <iostream.h>
 int k;
int *pk;
 char ch;
char *pch;
 double dArray[20];
 int main()
{
   cout << sizeof (int) << " - " << sizeof k << " - " << sizeof (pk) << endl;
      //      tip nomi            o'zgaruvchi              pointer
   cout <<sizeof (char) << " - " <<sizeof ch << " - " <<sizeof (pch) << endl;
    cout << "\nMassiv hotirada egallagan umumiy joy (baytlarda): "
        << sizeof (dArray) << endl;
   cout << "Massivning alohida elementi egallagan joy: "
        << sizeof (double) << endl;
   cout << "Massivdagi elementlar soni: "
        << sizeof (dArray) / sizeof (double) << endl;
    return (0);
}
```

Ekranda:

```
4 - 4 - 4
1 - 1 - 4

Massiv hotirada egallagan umumiy joy (baytlarda): 160
Massivning alohida elementi egallagan joy: 8
Massivdagi elementlar soni: 20
```