1 – DARS. C++ DASTURLASH TILINING KELIB CHIQISHI XAQIDA MA’LUMOT

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.

2 – DARS. TIL TUZILISHI.

1. Alfavit, identifikator, xizmatchi so'zlar.

Alfavit. C++ alfavitiga quyidagi simvollar kiradi.

- Katta va kichik lotin alfaviti xarflari (A,B,..,Z,a,b,...,z)
- Raqamlar: 0,1,2,3,4,5,6,7,8,9
- Maxsus simvollar: " , {} | [] () + - / % \ ; ' . : ? < = > _ ! & * # ~ ^
- Ko'rinmaydigan simvollar ("umumlashgan bushliq simvollari"). Leksemalarni uzaro ajratish uchun ishlatiladigan simvollar (misol uchun bushlik, tabulyatsiya, yangi qatorga o'tish belgilari).

Izohlarda, satrlarda va simvolli konstantalarda boshqa literalar, masalan rus xarflarini ishlatilishi mumkin.
C++ tilida olti khil turdagi leksemalar ishlatiladi: ehrkin tanlanadigan va ishlatiladigan identifikatorlar, khizmatchi suzlar, konstantalar(konstanta satrlar), amallar(amallar belgilari), azhratuvchi belgilar.

Identifikator.Identifikatorlar lotin xarflari,ostki chiziq belgisi va sonlar ketma ketligidan iborat buladi. Identifikator lotin xarfidan yoki ostki chizish belgisidan boshlanishi lozim.
Misol uchun:
A1, _MAX, adress_01, RIM, rim
Katta va kichik xarflar farklanadi, shuning uchun ohirgi ikki identifikator bir biridan farq qiladi.
Borland kompilyatorlaridan foydalanilganda nomning birinchi 32 xarfi ,ba'zi kompilyatorlarda 8 ta xarfi inobatga olinadi. Bu holda NUMBER_OF_TEST va NUMBER_OF_ROOM identifikatorlari bir biridan farq qilmaydi.

Xizmatchi so'zlar. Tilda ishlatiluvchi ya'ni dasturchi tomonidan uzgaruvchilar nomlari sifatida ishlatish mumkin bulmagan identifikatorlar xizmatchi so'zlar deyiladi.
C ++ tilida quyidagi xizmachi so'zlar mavjud:

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

3 – DARS. O’zgaruvchilar. (VARIABLES)

O’zgaruvchilar ob’ekt sifatida. Ci++ tilining asosiy tushunchalaridan biri nomlangan hotira qismi – ob’ekt tushunchasidir. Ob’ektning xususiy holi bu o’zgaruvchidir. O’zgaruvchiga qiymat berilganda unga ajratilgan hotira qismiga shu qiymat kodi yoziladi. O’zgaruvchi qiymatiga nomi orqali murojaat qilish mumkin, hotira qismiga esa faqat adresi orqali murojaat qilinadi. O’zgaruvchi nomi bu erkin kiritiladigan identifikatordor. O’zgaruvchi nomi sifatida xizmatchi so’zlarni ishlatish mumkin emas.

O’zgaruvchilar tiplari. O’zgaruvchilarning qo’yidagi tiplari mavjuddir:

char – bitta simvol;
long char – uzun simvol;
int – butun son;
short yoki short int – qisqa butun son;
long yoki long int – uzun butun son;
float - haqiqiy son;
long float yoki double – ikkilangan haqiqiy son;
long double – uzun ikkilangan haqiqiy son;

Butun sonlar ta’riflanganda ko’rilgan tiplar oldiga unsigned (ishorasiz) ta’rifi kushilishi mumkin. Bu ta’rif qushilgan butun sonlar ustida amallar mod 2n arifmetikasiga asoslangandir . Bu erda n soni int tipi hotirada egallovchi razryadlar sonidir. Agar ishorasiz k soni uzunligi int soni razryadlar sonidan uzun bulsa, bu son qiyjmati k mod 2n ga teng bo'ladi. Ishorasiz k son uchun ga –k amali 2n – k formula asosida hisoblanadi. Ishorali ya’ni signed tipidagi sonlarning eng katta razryadi son ishorasini ko’rsatish uchun ishlatilsa unsigned (ishorasiz) tipdagi sonlarda bu razryad sonni tasvirlash uchun ishlatiladi.

O’zgaruvchilarni dasturning ihtiyoriy qismida ta’riflash yoki qayta ta’riflash mumkin.
Misol uchun:

```
Int a, b1, ac; eki
Int a;
int b1;
int ac;
```

O’zgaruvchilar ta’riflanganda ularning qiymatlari aniqlanmagan bo’ladi. Lekin o’zgaruvchilarni ta’riflashda initsializatsiya ya’ni boshlang’ich qiyjmatlarini ko’rsatish mumkin.
Misol uchun:

```
Int I=0;
Char c=’k’;
```

Typedef ta’riflovchisi yangi tiplarni kiritishga imkon beradi.
Misol uchun yangi COD tipini kiritish:

```
Typedef unsigned char COD;
COD simbol;
```

4 – DARS. KONSTANTALAR. (CONSTANTS)

Konstanta bu o'zgartirish mumkin bulmagan qiymatdir. C++ tilida besh turdagi konstantalar ishlatilishi mumkin: butun sonlar, haqiqiy sonlar, simvollar, sanovchi konstantalar va nul kursatkich.

1. Ma'lumotlarning butun son turi.

Butun sonlar o'nlik, sakkizlik yoki un oltilik sanoq sistemalarida berilishi mumkin.
O'nlik sanoq sistemasida butun sonlar 0-9 raqamlari ketma ketligidan iborat bo'lib, birinchi raqami 0 bulishi kerak emas.
Sakkizlik sanoq sistemasida butun sonlar 0 bilan boshlanuvchi 0-7 raqamlaridan iborat ketma ketlikdir.
O'n oltilik sanoq sistemasida butun son 0x eki 0X bilan boshlanuvchi 0-9 raqamlari va a-f yoki A-F xarflaridan iborat ketma ketlikdir.
Masalan 15 va 22 o'nlik sonlari sakkizlikda 017 va 026, un oltilikda 0xF va 0x16 shaklda tasvirlanadi.
Ma'lumolarning uzun butun son turi.
Oxiriga l eki L harflari quyilgan o'nlik,sakkizlik yoki o'n oltilik butun son.
Ma'lumotlarning ishorasiz (unsigned) butun son turi:
Ohiriga u yoki U harflari quyilgan o'nlik,sakkizlik yoki o'n oltilik oddiy yoki uzun butun son.
2. Ma'lumotlarning haqiqiy son turi:
Olti qismdan iborat bulishi mumkin: butun qism, nuqta, kasr qism, yoki E belgisi, o'nlik daraja , F eki f suffikslari.
Masalan : 66. .0 .12 3.14F 1.12e-12
Ma'lumolarning uzun haqiqiy son turi :
Ohiriga L eki l suffikslari quyjilgan haqiqiy son.
Masalan: 2E+6L;
3. Simvolli konstanta.
Bittalik qavslarga olingan bitta yoki ikkita simvol. Misol uchun 'x','*','\012','\0','\n'- bitta simvolli konstanta; 'dd','\n\t','\x07\x07' ikki simvolli konstantalar.
'\' simvolidan boshlangan simvollar eskeyp simvollar deyjiladi.Simvolli konstanta qiymati simvolning kompyuterda qabul qilingan sonli kodiga tengdir.

ESC (eskeyp) simvollar jadvali:

Yozilishi	Ichki kodi	Simvoli (nomi)	Ma'nosi
\a	0x07	bel (audible bell)	Tovush signali
\b	0x08	Bs (bascspase)	Bir qadam qaytish
\f	0x0C	Ff (form feed)	Sahifani qaytarish
\n	0x0A	lf (line feed)	Qatorni o'tkazish
\r	0x0D	Cr (carriage return)	Karetkani qaytarish
\t	0x09	Ht (horizontal tab)	Gorizontal tabulyatsi
\v	0x0B	Vt (vertical tab)	Vertikal tabulyatsi
\\	0x5C	\ (bacslash)	Teskari chiziq
\'	0x27	' (single out)	Apostrif (oddiy qavs)
\"	0x22	" (double quote)	Ikkilik qavs
\?	0x3F	? (question mark)	Savol Belgisi
\000	000	Ëþ áî é (octal number)	Simvol sakkizlik kodi
\xhh	0xhh	Ëþ áî é (hex number)	Simvol o'n oltilik kodi

Satrli konstanta.

Satrli konstantalar C++ tili konstantalariga kirmaydi, balki leksemalari alohida tipi hisoblanadi. Shuning uchun adabiyotda satrli konstantalar satrli leksemalar deb ham ataladi..

Satrli konstanta bu ikkilik qavslarga olingan ihtiyoriy simvollar ketma ketligidir. Misol uchun " Men satrli konstantaman".

Satrlar orasiga eskeyp simvollar ham kirishi mumkin. Bu simvollar oldiga \ belgisi quyiladi. Misol uchun :

"\n Bu satr \n uch katorga \n zhoyjlashadi".

Satr simvollari hotirada ketma-ket joylashtiriladi va har bir satrli konstanta ohiriga avtomatik ravishda kompilyator tomonidan '\0' simvoli qo'shiladi. Shunday satrning hotiradagi hazhmi simvollar soni+1 baytga tengdir.

Ketma-ket kelgan va bushlik, tabulyatsiya yoki satr ohiri belgisi bilan ajratilgan satrlar kompilyatsiya davrida bitta satrga aylantiriladi. Misol uchun:

"Salom" "Toshkent "
satrlari bitta satr deb qaraladi.
"Salom Toshkent"

Bu qoidaga bir necha qatorga yozilgan satrlar ham buysinadi. Misol uchun :

"O'zbekistonga "
"bahor "
"keldi"

qatorlari bitta qatorga mos:

"O'zbekistonga bahor keldi"

Agar satrda '\' belgisi uchrasa va bu belgidan so'ng to '\n' satr ohiri belgisigacha bushlik belgisi kelsa bu bushlik belgilari '\' va '\n' belgisi bilan birga satrdan uchiriladi. Satrning uzi keyingi satrda kelgan satr bilan qo'shiladi.

"Ozbekistonga \
" bahor\
" keldi"

qatorlari bitta qatorga mos:

"Uzbekistonga bakhor keldi"

Sanovchi konstanta.

Sanovchi konstantalar enum hizmatchi so'zi yordamida kiritilib, int tipidagi sonlarga qulay suzlarni mos quyish uchun ishlatiladi.

Misol uchun:

```
enum{one=1,two=2,three=3};
```

Agar son qiymatlari ko'rsatilmagan bulsa eng chapki so'zga 0 qiymati berilib qolganlariga tartib buyicha usuvchi sonlar mos quyiladi:

```
Enum{zero,one,two};
```

Bu misolda avtomatik ravishda konstantalar quyidagi qiymatlarni qabul qiladi:

```
Zero=0, one=1, two=2;
```

Konstantalar aralash ko'rinishda kiritilishi ham mumkin:

```
Enum(zero,one,for=4,five,seeks}.
```

Bu misolda avtomatik ravishda konstantalar quyidagi qiyjmatlarni qabul qiladi:

```
Zero=0, one=1, for=4;five=5,seeks=6;
```

Yana bir misol:

```
Enum BOOLEAN {NO, YES};
```

Konstantalar qiymatlari:
NO=0, YES=1;

Nomlangan konstantalar.

CI ++ tilida o'zgaruvchilardan tashqari nomlangan konstantalar kiritilishi mumkin. Bu konstantalar qiymatlarini dasturda o'zgartirish mumkin ehmas. Konstantalar nomlari dasturchi tomonidan kiritilgan va hizmatchi so'zlardan farqli bo'lgan identifikatorlar bulishi mumkin. Odatda nom sifatida katta lotin harflari va ostiga chizish belgilari kombinaciyasidan iborat identifikatorlar ishlatiladi. Nomlangan konstantalar quyidagi shaklda kiritiladi:
Const tip konstanta_nomi=konstanta_kiyjmati.
Misol uchun:
Const double EULER=2.718282;
Const long M=99999999;
Const R=765;
Ohirgi misolda konstanta tipi kursatilmagan, bu konstanta int tipiga tegishli deb hisoblanadi.

Nul ko'rsatkich.

NULL- ko'rsatkich yagona arifmetik bulmagan konstantadir. Konkret realizatsiyalarda null ko'rsatkich 0 eki 0L eki nomlangan konstanta NULL orqali tasvirlanishi mumkin. Shuni aytish lozimki bu konstanta qiymati 0 bo'lishi eki '0' simvoli kodiga mos kelishi shart ehmas.

Quyidagi jadvalda konstantalar chegaralari va mos tiplari ko'rsatilgan:

Ma'lumotlar turi	Hajm, bit	Qiymatlar chegarasi	Tip vazifasi
Unsigned char	8	0...255	Kichik butun sonlar va simvollar kodlari
Char	8	-128...127	Kichik butun sonlar va ASII kodlar
Enum	16	-32768...32767	Butun sonlar tartiblangan katori
Unsigned int	16	0...65535	Katta butun sonlar
Short int	16	-32768...32767	Kichik butun sonlar, tsikllarni boshqarish
Int	16	-32768...32767	Kichik butun sonlar, tsikllarni boshqarish
Unsigned long	32	0...4294967295	Astronomik masofalar
Long	32	-147483648... ...2147483647	Katta sonlar
Float	32	3.4E-32...3.4E+38	Ilmiy hisoblar (7 raqam)
Double	64	1.7E-308...1.7E+308	Ilmiy hisoblar(15 raqam)
Long double	80	3.4E-4932... 1.1E+4932	Moliyaviy hisobalr (19 raqam)

5 – DARS. C++ DA AMALLAR

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,	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 ñhiqadigan 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'nosi 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 ) );
```

Amallar jadvali

Arifmetik amallar		Razryadli amallar		Nisbat amallari		Mantiqiy amallar	
+	qo'shish	&	va	= =	teng	&&	va
-	bo'lish	|	yoki	!=	teng emas	||	yoki
*	ko'paytirish	^	inkor	>	katta	!	inkor
/	bo'lish	<<	chapga surish	>=	katta yoki teng		
%	modul olish	>>	o'ngga surish	<	kichik		
-	unar minus	~	inkor	<=	kichik yoki teng		
+	unar plyus						
++	oshirish						
--	kamaytirish						

Amallar jadvali (davomi)

Imlo amallar	Qiymat berish va shartli amallar	Tipli amallar	Adresli amallar
() – doirali qavs	= - oddiy qiymar berish	(tip) – tipni o'zgartirish	& - adresni aniqlash
[] – kavadrat qavs	op= - murakkab qiymat berish	sizeof- hajmni hisoblash	* - adres bo'yicha qiymat aniqlash yoki joylash
, - vergul	? – shartli amal		

Arifmetik amallar. Amallar odatda unar ya'ni bitta operandga qo'llaniladigan amallarga va binar ya'ni ikki operandga qo'llaniladigan amallarga ajratiladi.
Binar amallar additiv ya'ni + qo'shuv va – ayirish amallariga , hamda multiplikativ ya'ni * kupaytirish, / bulish va % modul olish amallariga ajratiladi.
Additiv amallarining ustivorligi multiplikativ amallarining ustivorligidan pastroqdir.
Butun sonni butun songa bo'lganda natija butun songacha yahlitlanadi. Misol uchun 20/3=6; (-20)/3=-6; 20/(-3)=-6.
Modul amali butun sonni butun songa bulishdan hosil buladigan qoldikka tengdir. Agar modul amali musbat operandlarga qo'llanilsa, natija ham musbat bo'ladi, aks holda natija ishorasi kompilyatorga bog'likdir.
Binar arifmetik amallar bajarilganda tiplarni keltirish quyidagi qoidalar asosida amalga oshiriladi:
short va char tiplari int tipiga keltiriladi;
Agar operandlar biri long tipiga tegishli bo'lsa ikkinchi operand ham long tipiga keltiriladi va natija ham long tipiga tegishli buladi;
Agar operandlar biri float tipiga tegishli bulsa ikkinchi operand kham float tipiga keltiriladi va natija ham float tipiga tegishli buladi;
Agar operandlar biri double tipiga tegishli bo'lsa ikkinchi operand ham double tipiga keltiriladi va natija ham double tipiga tegishli buladi;
Agar operandlar biri long double tipiga tegishli bo'lsa ikkinchi operand ham long double tipiga keltiriladi va natija ham long double tipiga tegishli bo'ladi;
Unar amallarga ishorani o'zgartiruvchi unar minus – va unar + amallari kiradi. Bundan tashqari ++ va -- amallari ham unar amallarga kiradi.
++ unar amali qiymatni 1 ga oshirishni ko'rsatadi. Amalni prefiks ya'ni ++i ko'rinishda ishlatish oldin o'zgaruvchi qiymatini oshirib so'ngra foydalanish lozimligini, postfiks ya'ni i++ ko'rinishda ishlatish oldin o'zgaruvchi qiymatidan foydalanib so'ngra oshirish kerakligini ko'rsatadi. Misol uchun i qiymati 2 ga teng bo'lsin, u holda 3+(++i) ifoda qiymati 6 ga, 3+i++ ifoda qiymati 5 ga teng bo'ladi. Ikkala holda ham i qiymati 3 ga teng bo'ladi.
-- unar amali qiymatni 1 ga kamaytirishni ko'rsatadi. Bu amal ham prefiks va postfiks ko'rinishda ishlatilishi mumkin. Bu ikki amalni faqat o'zgaruvchilarga qo'llash mumkindir.
Unar amallarning ustivorligi binar amallardan yuqoridir.
Razryadli amallar. Razryadli amallar natijasi butun sonlarni ikkilik ko'rinishlarining har bir razryadiga mos mantikiy amallarni qo'llashdan hosil bo'ladi. Masalan 5 kodi 101 ga teng va 6 kodi 110 ga teng.

6&5 qiyjmati 4 ga ya'ni 100 ga teng.
6|5 qiyjmati 7 ga ya'ni 111 ga teng.
6^5 qiymati 3 ga ya'ni 011 ga teng.

~6 kiyjmati 4 ga yajhni 010 ga teng.
Bu misollarda amallar ustivorligi oshib borishi tartibida berilgandir.
Bu amallardan tashqari M<<N chapga razryadli siljitish va M>>N ungga razryadli siljitish amallari qo'llaniladi. Siljitish M butun sonning razryadli ko'rinishiga qo'llaniladi. N nechta pozitsiyaga siljitish kerakligini ko'rsatadi.
Chapga N pozitsiyaga surish bu operand qiymatini ikkining N chi daraasiga kupaytirishga mos keladi. Misol uchun 5<<2=20. Bu amalning bitli kurinishi: 101<<2=10100.
Agar operand musbat bulsa N poziciyaga ungga surish chap operandni ikkining N chi darajasiga bo'lib kasr qismini tashlab yuborishga mosdir. Misol uchun 5>>2=1. Bu amalning bitli kurinishi 101>>2=001=1. Agarda operand qiymati manfiy bulsa ikki variant mavjuddir: arifmetik siljitishda bushatilayotgan razryadlar ishora razryadi qiymati bilan to'ldiriladi, mantiqiy siljitishda bushatilayotgan razryadlar nullar bilan tuldiriladi.
Razryadli surish amallarining ustivorligi o'zaro teng, razryadli inkor amalidan past, qolgan razryadli amallardan yuqoridir. Razryadli inkor amali unar qolgan amallar binar amallarga kiradi.
Nisbat amallari. Nisbat amallari qiymatlari 1 ga teng agar nisbat bajarilsa va aksincha 0 ga tengdir. Nisbat amallari arifmetik tipdagi operandlarga yoki ko'rsatkichlarga qo'llaniladi.
Misollar:
1!=0 qiymati 1 ga teng;
1==0 qiymati 0 ga teng;
3>=3 qiymati 1 ga teng;
3>3 qiymati 0 ga teng;
2<=2 qiymati 1 ga teng;
2<2 qiymati 0 ga teng;
Katta >, kichik <, katta eki teng >=, kichik eki teng <= amallarining ustivorligi bir hildir.
Teng == va teng emas != amallarining ustivorligi uzaro teng va qolgan amallardan pastdir.
Mantiqiy amallar. C ++ tilida mantiqiy tip yukdir. Shuning uchun mantiqiy amallarni butun sonlarga qo'llanadi. Bu amallarning natijalari qo'yidagicha aniqlanadi:
x||y amali 1 ga teng agar x>0 eki y>0 bo'lsa, aksincha 0 ga teng
x&&y amali 1 ga teng agar x>0 va y>0 bo'lsa, aksincha 0 ga teng
 !x amali 1 ga teng agar x>0 bulsa, aksincha 0 ga teng
Bu misollarda amallar ustivorligi oshib borish tartibida berilgandir.
Inkor ! amali unar kolganlari binar amallardir.
Bu amallardan tashqari quyidagi amallar ham mavjuddir:
Qiymat berish amali. Qiymat berish amali = binar amal bo'lib chap operandi odatda o'zgaruvchi ung operandi odatda ifodaga teng bo'ladi. Misol uchun Z=4.7+3.34
Bu qiymati 8.04 ga teng ifodadir. Bu qiymat Z o'zgaruvchiga ham beriladi.
Bu ifoda ohiriga nuqta vergul ; belgisi quyilganda operatorga aylanadi.
Z=4.7+3.34
Bitta ifodada bir necha qiymat berish amallari qo'llanilishi mumkin. Misol uchun:
C=y=f=4.2+2.8;
Bundan tashqari C ++ tili da murakkab qiymat berish amali mavjud bo'lib, umumiy ko'rinishi quyidagichadir:
O'zgaruvchi_nomi amal= ifoda;
Bu erda amal quyidagi amallardan biri *,/,%,+,-, &,^,|, <<,>>.
Misol uchun:

X+=4 ifoda x=x+4 ifodaga ekvivalentdir;
X*=a ifoda x=x*a ifodaga ekvivalentdir;
X/=a+b ifoda x=x/(a+b) ifodaga ekvivalentdir;
X>>=4 ifoda x=x>>4 ifodaga ekvivalentdir;
Imlo belgilari amal sifatida. C ++ tilida ba'zi bir imlo belgilari ham amal sifatida ishlatilishi mumkin. Bu belgilardan oddiy () va kvadrat [] qavslardir. Oddiy qavslar binar amal deb qaralib ifodalarda yoki funktsiyaga muroaat qilishda foydalaniladi. Funktsiyaga murojaat qilish qo'yjidagi shaklda amalga oshiriladi:
<funktsiya nomi> (<argumentlar ruyhati>). Misol uchun sin(x) eki max(a,b).
Kvadrat qavslardan massivlarga murojaat qilishda foydalaniladi. Bu murojaat quyidagicha amalga oshiriladi:
<massiv nomi>[<indeks>]. Misol uchun a[5] eki b[n][m].
Vergul simvolini ajratuvchi belgi deb ham qarash mumkin amal sifatida ham qarash mumkin. Vergul bilan ajratilgan amallar ketma-ketligi bir amal deb qaralib, chapdan o'ngga hisoblanadi va ohirgi ifoda qiymati natija deb qaraladi. Misol uchun:
d=4,d+2 amali natijasi 8 ga teng.
Shartli amal. Shartli amal ternar amal deyiladi va uchta operanddan iborat bo'ladi:
<1-ifoda>?<2-ifoda>:<3-ifoda>
Shartli amal bajarilganda avval 1- ifoda hisoblanadi. Agar 1-ifoda qiymati 0 dan farqli bo'lsa 2- ifoda hisoblanadi va qiymati natija sifatida qabul qilinadi, aks holda 3-ifoda hisoblanadi va qiymati natija sifatida qabul qilinadi.
Misol uchun modulni hisoblash: x<0?-x:x yoki ikkita son kichigini hisoblash a<b?a:b .
Shuni aytish lozimki shartli ifodadan har qanday ifoda sifatida foydalanish mumkin. Agar F FLOAT tipga,a N – INT tipga tegishli bo'lsa ,
(N > 0) ? F : N
ifoda N musbat eki manfiyligidan qat'iy nazar DOUBLE tipiga tegishli bo'ladi.
Shartli ifodada birinchi ifodani qavsga olish shart emas.
Tiplar bilan ishlovchi amallar. Tiplarni o'zgartirish amali quyidagi ko'rinishga ega:
(tip_nomi) operand; Bu amal operandlar qiymatini ko'rsatilgan tipga keltirish uchun ishlatiladi. Operand sifatida kostanta, o'zgaruvchi yoki qavslarga olinga ifoda kelishi mumkin. Misol uchun (long)6 amali konstanta qiymatini o'zgartirmagan holda operativ hotirada egallagan baytlar sonini oshiradi. Bu misolda konstanta tipi o'zgarmagan bo'lsa, (double) 6 eki (float) 6 amali konstanta ichki ko'rinishini ham o'zgartiradi. Katta butun sonlar hakikiy tipga keltirilganda sonning aniqligi yuqolishi mumkin.
sizeof amali operand sifatida ko'rsatilgan ob'ektning baytlarda hotiradagi hajmini hisoblash uchun ishlatiladi. Bu amalning ikki ko'rinishi mavjud:
sizeof ifoda sizeof (tip) Misol uchun:

Sizeof 3.14=8
Sizeof 3.14f=4
Sizeof 3.14L=10
Sizeof(char)=1
Sizeof(double)=8.

Amallar ustivorligi

Rang	Amallar	Yo'nalish
1	() [] -> :: .	Chapdan o'ngga
2	! ~ + - ++ -- & * (tip) sizeof new delete tip()	O'ngdan chapga
3	. * ->*	Chapdan o'ngga
4	* / % (multiplikativ binar amallar)	Chapdan o'ngga
5	+ - (additiv binar amallar)	Chapdan o'ngga
6	<< >>	Chapdan o'ngga
7	< <= >= >	Chapdan o'ngga
8	= !=	Chapdan o'ngga
9	&	Chapdan o'ngga
10	^	Chapdan o'ngga
11	|	Chapdan o'ngga
12	&&	Chapdan o'ngga
13	||	Chapdan o'ngga
14	?:(shartli amal)	Chapdan o'ngga
15	= *= /= %= += -= &= ^= |= <<= >>=	Chapdan o'ngga
16	, (vergul amali)	Chapdan o'ngga

6 – DARS. DASTUR TUZILISHI.

Sodda dastur tuzilishi. Dastur preprocessor komandalari va bir necha funktsiyalardan iborat bo'lishi mumkin. Bu funktsiyalar orasida main nomli asosiy funktsiya bo'lishi shart. Agar asosiy funktsiyadan boshqa funktsiyalar ishlatilmasa dastur quyidagi ko'rinishda tuziladi:

```
Preprocessor_komandalari
Void main()
{
Dastur tanasi.
}
```

Preprocessor direktivalari kompilyatsiya jarayonidan oldin preprocessor tomonidan bajariladi. Natijada dastur matni preprocessor direktivalari asosida o'zgartiriladi. Preprocessor komandalaridan ikkitasini ko'rib chiqamiz.

include <fayl_nomi> Bu direktiva standart bibliotekalardagi funktsiyalarni dasturga joyjlash uchun foydalaniladi.

#define <almashtiruvchi ifoda> <almashinuvchi ifoda>

Bu direktiva bajarilganda dastur matnidagi almashtiruvchi ifodalar almashinuvchi ifodalarga almashtiriladi.

Misol tariqasida C ++ tilida tuzilgan birinchi dasturni keltiramiz:

```
#include <iostream.h>
void main()
{
   Cout << "\n Salom, Dunyo!  \n";
}
```

Bu dastur ehkranga Salom, Dunyo! Jumlasini chiqaradi.

Define direktivasi yordamida bu dasturni quyidagicha yozish mumkin:

```
#include <iostream.h>
#define pr      Cout << "\n Salom, Dunyo! \n"
#define begin {
#define end  }
void main()
  begin
    pr;
  end
```

Define direktivasidan nomlangan konstantalar kiritish uchun foydalanish mumkindir. Misol uchun:

```
#define EULER 2.718282
```

Agar dasturda quyidagi matn mavjud bo'lsin:

```
Double mix=EULER
D=alfa*EULER
```

Preprocessor bu matnda har bir EULER konstantani uning qiymati bilan almashtiradi, va natijada quyidagi matn hosil bo'ladi.

```
Double mix=2.718282
D=alfa*2.718282
```

Dastur matni va preprocessor. C ++ tilida matnli fayl shaklida tayyorlangan dastur uchta qayta ishlash bosqichlaridan o'tadi.
Matnni preprocessor direktivalari asosida o'zgartilishi. Bu jarayon natijasi Yana matnli fayl bo'lib preprocessor tomonidan bajariladi.
Kompilyatsiya. Bu jarayon natijasi mashina kodiga o'tkazilgan obektli fayl bo'lib, kompilyator tomonidan bajariladi.
Bog'lash. Bu jarayon natijasi to'la mashina kodiga o'tkazilgan bajariluvchi fayl bo'lib, boglagich(komponovthik) tomonidan bajariladi.
Preprocessor vazifasi dastur matnini preprocessor direktivalari asosida o'zgartirishdir. Define direktivasi dasturda bir jumlani ikkinchi jumla bilan almashtirish uchun ishlatiladi. Bu direktivadan foydalanishning sodda misollarini biz yuqorida ko'rib chiqdik. Include direktivasi ikki ko'rinishda ishlatilishi mumkin.
#include fayl nomi direktivasi dasturning shu direktiva urniga qaysi matnli fayllarni qo'shish kerakligini ko'rsatadi.
#include <fayl nomi> direktivasi dasturga kompilyator standart bibliotekalariga mos keluvchi sarlavhali fayllar matnlarini qushish uchun muljhallangandir. Bu fayllarda funktsiya prototipi, tiplar, o'zgaruvchilar, konstantalar ta'riflari yozilgan buladi. Funktsiya prototipi funktsiya qaytaruvchi tip, funktsiya nomi va funktsiyaga uzatiluvchi tiplardan iborat bo'ladi. Misol uchun cos funkciyasi prototipi quyidagicha yozilishi mumkin: double cos(double). Agar funkciya nomidan oldin void tipi ko'rsatilgan bo'lsa bu funktsiya hech qanday qiymat qaytarmasligini ko'rsatadi.Shuni ta'kidlash lozimki bu direktiva dasturga standart biblioteka qo'shilishiga olib kelmayjdi. Standart funktsiyalarning kodlari bog'lash ya'ni aloqalarni tahrirlash bosqichida, kompilyatsiya bosqichidan so'ng amalga oshiriladi.
Kompilyatsiya bosqichida sintaksis hatolar tekshiriladi va dasturda bunday hatolar mavjud bo'lmasa, standart funktsiyalar kodlarisiz mashina kodiga utkaziladi.
Sarlavhali fayllarni dasturning ihtiyoriy joyida ulash mumkin bo'lsa ham, bu fayllar odatda dastur boshida qo'shish lozimdir. Shuning uchun bu fayllarga sarlavhali fayl (header file) nomi berilgandir.
Dasturda kiritish va chiqarish funktsiyalaridan masalan Cout<< funktsiyasidan foydalanish uchun #include <iostream.h> direktivasidan foydalanish lozimdir Bu direktivada iostream.h sarlavhali fayl nomi quyidagilarni bildiradi: st- standart(standartnij), i- input(vvod), o- output(vihvod), h – head(sarlavha).

7 – Dars. 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.

8 – DARS. KO'RSATKICHLAR. (POINTER)

Ko'rsatkichlar ta'rifi. C va C++ tillarining asosiy hususiyatlaridan ko'rsatkichlarning keng qo'llanilishidir. Ko'rsatkichlar tilda konstanta ko'rsatkichlar va o'zgaruvchi ko'rsatkichlarga ajratiladi. Ko'rsatkichlar qiymati konkret tipdagi ob'ektlar uchun hotirada ajratilgan adreslarga tengdir. Shuning uchun ko'rsatkichlar ta'riflanganda ularning adreslarini ko'rsatish shart. O'zgaruvchi ko'rsatkichlar qo'yidagicha ta'riflanadi.

<tip> * <ko'rsatkich nomi>

Misol uchun int* lp,lk .

Ko'rsatkichlarni ta'riflaganda initsializatsiya qilish mumkindir. Initsializatsiya quyidagi shaklda amalga oshiriladi:

<tip> * <ko'rsatkich nomi>=<konstanta ifoda>

Konstanta ifoda sifatida qo'yidagilar kelishi mumkin.

- Hotira qismining aniq ko'rsatilgan adresi. Misol uchun:

char* comp=(char*) 0xF000FFFE; Bu adresda kompyuter tipi shaklidagi ma'lumot saqlanadi.

- Qiymatga ega ko'rsatkich: char c1=comp;
- & simvoli yordamida aniqlangan ob'ekt adresi. Misol uchun:

  char c='d'; char* pc=&c;

Borland kompilyatorlarida mahsus NULL kiymat kiritilgan bulib, bu qiymatga e'ga ko'rsatkichlar bush ko'rsatkichlar deyiladi. Bush ko'rsatkichlar bilan hotirada hech qanday adres bog'lanmagan bo'ladi, lekin dasturda konkret obektlar adreslarini qiyjmat sifatida berish mumkin.

Char ca='d'; char* pa(NULL); pa=&ca;

Ko'rsatkichlar ustida amallar. Yuqorida keltirilgan misollarda & adres olish amalidan keng foydalanilgan. Bu amal nomga va hotirada aniq adresga ega ob'ektlarga, misol uchun o'zgaruvchilarga qo'llaniladi. Bu amalni ifodalarga eki nomsiz konstantalarga qo'llash mumkin emas. Ya'ni &3.14 eki &(a+b) ifodalar hato hisoblanadi.

Bundan tashqari ko'rsatkichlar bilan birga * adres buyjicha kiymat olish eki kiritish amali keng qullaniladi. Misol uchun:

Int i=5; int*pi=&I; int k=*pi; *pi=6.

Bu misolda pi kursatkich I uzgaruvchi bilan boglanadi. *pi=6 amali I uzgaruvchi qiymatini ham uzgartiradi.

Konstanta ko'rsatkich va konstantaga ko'rsatkichlar. Konstanta ko'rsatkich quyidagicha ta'riflanadi:

<tip>* const<kursatkich nomi>=<konstanta ifoda>

Misol uchun: char* const key_byte=(char*)0x0417. Bu misolda konstanta ko'rsatkich klaviatura holatini ko'rsatuvchi bayt bilan bog'langandir.

Konstanta ko'rsatkich qiymatini o'zgartirish mumkin emas lekin * amali yordamida hotiradagi ma'hlumot qiymatini o'zgartirish mumkin. Misol uchun *key_byte='Yo' amali 1047(0x0417) adres qiymati bilan birga klaviatura holatini ham oz'zgartiradi.

Konstantaga ko'rsatkich quyidagicha ta'riflanadi:

<tip>const*<kursatkich nomi>=<konstanta ifoda>. Misol uchun const int zero=0; int const* p=&zero;

Bu ko'rsatkichga * amalini qullash mumkin emas, lekin ko'rsatkichning qiyjmatini o'zgartirish mumkin. Qiymati o'zgarmaydigan konstantaga ko'rsatkichlar quyidagicha kiritiladi:

<tip>const* const<kursatkich nomi>=<konstanta ifoda>. Misol uchun const float pi=3.141593; float const* const pp=π

9 – DARS. OPERATORLAR VA BLOKLAR.

Har qanday dastur funktsiyalar ketma ketligidan iborat bo'ladi. Funktsiyalar sarlavha va funktsiya tanasidan iborat bo'ladi. Funktsiya sarlavhasiga void main() ifoda misol bo'la oladi. Funktsiya tanasi ob'ektlar ta'riflari va operatorlardan iborat bo'ladi.

Har qanday operator nuqta-vergul belgisi bilan tugashi lozim. Quyidagi ifodalar X=0, yoki I++ operatorga aylanadi agar ulardan so'ng nuqtali vergul kelsa

X = 0; I++;

Operatorlar bajariluvchi va bajarilmaydigan operatorlarga ajratiladi. Bajarilmaydigan operator bu izoh operatoridir.

Izoh operatori /* belgisi bilan boshlanib */ belgisi bilan tugaydi. Bu ikki simvol orasida ihtiyoriy jumla yozish mumkin. Kompilyator bu jumlani tekshirib o'tirmayjdi. Izoh operatoridan dasturni tushunarli qilish maqsadida izohlar kiritish uchun foydalaniladi.

Bajariluvchi operatorlar o'z navbatida ma'lumotlarni o'zgartiruvchi va boshqaruvchi operatorlarga ajratiladi.

Ma'lumotlarni o'zgartiruvchi operatorlarga qiymat berish operatorlari va nuqta vergul Bilan tugovchi ifodalar kiradi. Misol uchun:

```
I++;
X*=I;
I=x-4*I;
```

Boshqaruvchi operatorlar dasturni boshqaruvchi konstruktsiyalar deb ataladi. Bu operatorlarga quyidagilar kiradi:

Qo'shma operatorlar;
Tanlash operatorlari;
Tsikl operatorlari;
O'tish operatorlari;

Qo'shma operatorlar. Bir necha operatorlar { va } figurali qavslar yordamida qo'shma operatorlarga yoki bloklarga birlashtirilishi mumkin. Blok eki qo'shma operator sintaksis jihatdan bitta operatorga ekvivalentdir. Blokning qo'shma operatordan farqi shundaki blokda obektlar ta'riflari mavjud bo'lishi mumkin.

Quyidagi dastur qismi qo'shma operator:

```
{
n++;
summa+=(float)n;
}
```

Bu fragment bo'lsa blok:

```
{
int n=0;
n++;
summa+=(float)n;
}
```

Kiritish chiqarish operatorlari.

Chiquvchi oqim cout kelishilgan buyicha ekranga mos keladi. Lekin mahsus operatorlar yordamida oqimni printer eki faylga mos quyish mumkin. Misol uchun MS-DOS qo'yidagi komandasi FIRST.EXE dasturi chiqimshini printerga yunaltiradi:

S:\> FIRST > PRN <ENTER>

Quyidagi dastur 1001.SRR 1001 sonini ekranga chiqaradi:

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

```
  cout << 1001;
}
```
Dastur bajarilishi natijasi : S:\> 1001 <ENTER>
1001
Bir necha qiymatlarni chiqarish:
```
#include <iostream.h>
void main(void)
(
  cout << 1 << 0 << 0 << 1;
}
```
Natija:
S:\> 1001TOO <ENTER>
1001

10 – DARS. TANLASH OPERATORLARI.

Shartli operator. Shartli operator ikki ko'rinishda ishlatilishi mumkin:

```
If (ifoda)
1- operator
Else
2- operator
 eki
If (ifoda)
1-operator
```

Shartli operator bajarilganda avval ifoda hisoblanadi ; agar qiymat rost ya'ni nol'dan farqli bo'lsa 1- operator bajariladi. Agar qiymat yolg'on ya'ni nol' bo'lsa va else ishlatilsa 2-operator bajariladi. Else qism har doim eng yaqin if ga mos qo'yiladi.

```
if( n>0)
    if(a>b)
        Z=a;
    else
       Z=b;
```

Agar else qismni yuqori if ga mos quyish lozim bo'lsa, figurali qavslar ishlatish lozim.

```
if( n>0) {
    if(a>b)
        z=a;
}
else
       z=b;
```

Misol tariqasida uchta berilgan sonning eng kattasini aniqlash dasturini ko'ramiz:

```
#include <iostream.h>
void( )
 {    float a,b,c,max);
       Cout <<"\n a="; Cin>>a;
       Cout <<"\n b="; Cin>>b;
       Cout <<"\n c="; Cin>>c;
    if (a>b)
       if (a>c)  max=a else max=c;
    else
         if b>c then max=b else max=c;
     Cout <<"\n" <<max;
  }
```

Keyingi misolda kiritilgan ball va maksimal ball asosida baho aniqlanadi:

```
#include <iostream.h>
void main( )
 {    float ball,max_ball,baho);
       Cout<< "\n ball="; Cin>>("%f",&ball);
```

```
    Cout<<“\n max_ball=”; Cin>>max_ball;
    d=ball/max_ball;
  if (d>0.85) baho=5 else
    if (d>75) baho=4 else
    if (d>0.55) then baho=3 else baho=2;
   Cout<<“\n  baho;
}
```

Kalit bo’yicha tanlash operatori. Kalit bo’yicha o’tish switch operatori umumiy ko’rinishi qo’yidagicha

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

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

```
Include <iostream.h>
Int baho;
Cin>> baho;
Switch(baho)
{case 2:Cout <<“\n emon”;break;
 case 3:Cout <<“\n  urta”;break;
 case 4:Cout <<“\n  yahshi”;break;
 case 5:Cout <<“\n  a'lo”;break;
 default: Cout <<“\n baho notugri kiritilgan”;
};
}
```

Keyingi misolimizda kiritilgan simvol unli harf ekanligi aniqlanadi:

```
Include <iostream.h>
Int baho; Char c; Cin >> c;
Switch(c)
{case ‘a’:
case ‘u’:
case ‘o’:
case ‘i’:
Cout <<“\n Kiritilgan simvol unli harf”;break;
default: Cout <<“\n Kiritilgan simvol unli harf emas”;
};
}
```

11 – DARS.TSIKL OPERATORLARI.

While operatori. While operatori quyidagi umumiy ko'rinishga egadir:

```
While(ifoda)
Operator
```

Bu operator bajarilganda avval ifoda hisoblanadi. Agar uning qiymati 0 dan farqli bo'lsa operator bajariladi va ifoda qayta hisoblanadi. To ifoda qiymati 0 bo'lmaguncha tsikl qaytariladi.
Agar dasturda while (1); satr quyilsa bu dastur hech qachon tugamaydi.

Misol. Berilgan n gacha sonlar yigindisi.

```
Void main()
 {
long n,i=1,s=0;
cin >>n;
while (i<= n )
   s+=i++;
Cout<<"\n s="<< s;
      };
```

Bu dasturda s+=i++ ifoda s=s+i; i=i+1 ifodalarga ekvivalentdir.
Quyidagi dastur to nuqta bosilmaguncha kiritilgan simvollar va qatorlar soni hisoblanadi:

```
Void main()
{
long nc=0,nl=0;
char c='';
while (c!= '.' )
{++nc;
if (c =='\n') ++nl;
};
Cout<<("%1d\n", nc);
 Cout <<"\n satrlar="<< nl<<"simvollar="<< nc;
};
```

Do-While operatori. Do-While operatori umumiy ko'rinishi qo'yidagicha:

```
do
Operator
While(ifoda)
```

Tsikl operatorining bu ko'rinishida avval operator bajariladi so'ngra ifoda hisoblanadi. Agar uning qiymati 0 dan farqli bo'lsa operator yana bajariladi va hokazo. To ifoda qiymati 0 bo'lmaguncha tsikl qaytariladi.

Misol. Berilgan n gacha sonlar yigindisi.

```
Void main()
 {
long n,i=1,s=0;
cin >>n;
  do
```

```
  s+=i++;
while (i<= n );
Cout<<"\n s="<< s;
      };
```

Bu dasturning kamchiligi shundan iboratki agar n qiymati 0 ga teng eki manfiy bo'lsa ham, tsikl tanasi bir marta bajariladi va s qiymati birga teng bo'ladi.

Keyingi misolimizda simvolning kodini monitorga chiqaruvchi dasturni kuramiz. Bu misolda tsikl to ESC (kodi 27) tugmasi bosilmaguncha davom etadi. Shu bilan birga ESC klavishasining kodi ham ekranga chiqariladi.

```
# include <iostream.h>;
main ()
{
char d; int I;
do
cin>>d;
i=c;
Cout<<"\n "<<i;
while(i!=27);
};
```

For operatori. For operatori umumiy ko'rinishi qo'yidagicha:

```
For( 1-ifoda;2- ifoda; 3-ifoda)
Operator
```

Bu operator qo'yidagi operatorga mosdir.

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

Misol. Berilgan n gacha sonlar yigindisi.

```
# include <iostream.h>;
void main {
int n;
Cin>>n;
for(int i=1,s=0;i<=n; i++, s+=i);
Cout<<"\n",s;
};
```

FOR operatori tanasi bu misolda bush, lekin C ++ tili grammatikasi qoidalari FOR operatori tanaga ega bo'lishini talab qiladi. Bush operatorga mos keluvchi nuqta vergul' shu talabni bajarishga hizmat qiladi.

Keyingi dasturda kiritilgan jumlada satrlar, so'zlar va simvollar sonini hisoblanadi.

```
# include <iostream.h>;
#define       yes  1
#define          no   0
void main()
{
int c, nl, nw, inword;
inword = no;
```

```
nl = nw = nc = 0;
for(char c='';c!='.';cin>> c)
{++nc;
if (c == '\n')
++nl;
if (c==' ' ||c=='\n' ||c=='\t')
inword = no;
else if (inword == no)
inword = yes;
++nw;
}
Cout <<"\n satrlar="<< nl<<"suzlar="<< nw<<"simvollar="<< nc;
 }
```

Programma har gal so'zning birinchi simvolini uchratganda, mos o'zgaruvchi qiymatini bittaga oshiradi. INWORD o'zgaruvchisi programma so'z ichida ekanligini kuzatadi. Oldiniga bu o'zgaruvchiga so'z ichida emas ya'ni NO qiymati beriladi. YES va NO simvolik konstantalardan foydalanish dasturni o'qishni engillashtiradi.
NL = NW = NC = 0 katori kuyidagi katorga mos keladi;
NC = (NL = (NW = 0));

switch operatori

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 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.

Do while takrorlash operatori

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.

12 – DARS. O'TISH OPERATORLARI.

Break operatori. Ba'zi hollarda tsikl bajarilishini ihtiyoriy joyda tuhtatishga tug'ri keladi. Bu vazifani break operatori bajarishga imkon beradi. Bu operator darhol tsikl bajarilishini to'htatadi va boshqaruvni tsikldan keyingi operatorlarga uzatadi.
Misol uchun o'quvchining n ta olgan baholariga qarab uning o'qish sifatini aniqlovchi dasturini ko'ramiz. Buning uchun dasturda o'quvchining olgan minimal bahosi aniqlanadi

```
        # include <iostream.h>
        void main()
        {
        int I,n,min,p;
          while (1)
                {Cout<<"Baholar soni="; Cin>>n;};
if (n>0) break;
Cout<<("Hato! n>0 bulishi lozim ! \n");
for (I=1,min=5; I<=n; I++)
{ cin >>p;
   if (p<2)||(p>5) {min=0; break};
  if (min>p) min=p;
}
if (p<2)||(p>5) cout break;
switch(min)
case 0:cout<<"Baho notugri kiritilgan";break;
case 2:cout<<"Talaba yomon o'qiydi";break;
case 3:cout<<"Talaba o'rtacha o'qiydi";break;
case 4:cout<<"Talaba yahshi o'qiydi";break;
case 5:cout<<"Talaba a'lo o'qiydi";break;
}
```

Biz misolda hato kiritilgan n qiymatdan saqlanish uchun while(1) tsikl kiritilgan. Agar n>0 bulsa Break operatori tsiklni tuhtatadi va dastur bajarilishi davom etadi. Agar kiritilayotgan baholar chegarada yotmasa min ga 0 qiymat berilib darhol tsikldan chiqiladi.

Continue operatori. Tsikl bajarilishiga ta'sir o'tkazishga imkon beradigan yana bir operator Continue operatoridir. Bu operator tsikl qadamini bajarilishini tuhtatib for va while da ko'rsatilgan shartli tekshirishga o'tkazadi.
Qo'yidagi misolda ketma-ket kiritilayotgan sonlarning faqat musbatlarining yig'indisini hisoblaydi. Sonlarni kiritish 0 soni kiritilguncha davom etadi.

```
# include <iostream.h>
void main()
{ double s, x;
int x;
Cout<<("\n 0 bilan tugallanuvchi sonlar katorini kiriting \n");
for (x=1.0; s=0.0; k=0; x!=0.0);
{ Cin>>("%lf", &x);
if (x<=0.0) continue;
k++; s+=x;
}
Cout<<"\n summa="<<s<<"musbat sonlar ="<<k;
}
```

O’tish operatori GO TO.

O’tish operatorining ko’rinishi:

Go to <identifikator>. Bu operator identifikator bilan belgilangan operatorga o’tish kerakligini ko’rsatadi.

Misol uchun goto A1;...;A1:y=5;

Strukturali dasturlashda Go to operatoridan foydalanmaslik maslahat beriladi. Lekin ba'zi hollarda o’tish operatoridan foydalanish dasturlashni osonlashtiradi.

Misol uchun bir necha tsikldan birdan chiqish kerak bo’lib qolganda , tug’ridan-tugri break operatorini qo’llab bo’lmaydi, chunki u faqat eng ichki tsikldan chiqishga imkon beradi.

Quyidagi misolda n ta qatorga n tadan musbat son kiritiladi. Agar n yoki sonlardan biri manfiy bo’lsa, kiritish qaytariladi:

```
# include <iostream.h>
int n, I, j, k;
M1: Cout<<“\n n=”; Cin>>n;
If (n<=0) { Cout<<“\n hato! n>0 bulishi kerak”;
Go to M1;} ;
M: Cout<<“x sonlarni kiriting \n”;
For (I=1; I<=10; I++) {Cout<<“\n I=”<< i;
For (j=1 ; j<=10; j++) {Cin>> k;
if (k<=0) goto M;}
}
```

Bu masalani GOTO operatorisiz hal qilish uchun qo’shimcha o’zgaruvchi kiritish lozimdir.

```
# include <iostream.h>
int n, I, j, k;
while 1 {
Cout<<“\n n=”; Cin>>n;
if (n>0) break;
Cout<<“\n hato! n>0 bulishi kerak”;
} ;

int M=0;
While M
{ M=0;
Cout<<“x sonlarni kiriting \n”;
For (I=1; I<=10; I++) {
If (M) break;
Cout<<(“\n I=%, i);
For (j=1 ; j<=10; j++) {Cin>>(“%f”, k);
if (k<=0) {M=1;break;}
}
```

13 – DARS. 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'ladi.
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.

14 – DARS. 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 << end;
```

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)

15 – DARS. FOR TAKRORLASH OPERATORI

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'llanilish 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.

16 – DARS. BOSHQARUV OPERATORIDA CONTINUE VA BREAK IFODALARINI QO'LLASH.

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 tsikli (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 kombinatsiya-
sini 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.

17 – DARS. 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.

18 – DARS. 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.

19 – DARS. 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

20 – DARS. 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:

```
4
```

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

21 – DARS. AMALIY MISOLLAR.

Rekurrent qatorlar. Rekurrent qator deb shunday qatorga aytiladiki bu qatorning n chi hadi n ning qiymatiga va qatorning oldingi elementlariga bog'lik buladi. Bu bog'liklikni aks ettiruvchi formula rekurrent formula deb ataladi. Misol uchun n! (faktorial) ya'ni n gacha sonlar kupaytmasini qo'yidagi rekurrent formula yordamida hisoblash mumkin:
S0=1,Sn=Sn-1*n
Bu formulaga asoslangan dasturning asosiy qismi qo'yidagicha yoziladi:
For(int s=1,i=1;i<=n;i++) s*=i;
Rekurrent qatorga yana bir misol Fibonachchi sonlari qatori bo'lib, bu qator qo'yidagi rekurrent formullalar asosida ifodalanadi:
S0=1, S1=1,Sn=Sn-1+ Sn-2
Berilgan n gacha bulgan Fibonachchi sonlarini hisoblash dasturi:

```
Main()
{
 while (1)
  { Cin>>("\n %i",&n);
     if (n>2) break;
     Cout<<(" n qiymati notug'ri kiritilgan");
   };
for(int S0=1,S1=1,i=3;i<=n;i++)
     { S=S0+S1;S0=S1;S1=S;Cout<<("\n i=%i S=%i",&i,&S);};
}
```

Cheksiz qatorlar. Matematikada odatda biror qiymatni hisoblash shu miqdorga cheksiz yaqinlashuvchi qator hadlarini hisoblashga olib keladi. Amalda cheksiz qator hadlarini hisoblash yaqinlashish sharti bajarilguncha davom etadi va bu shartga mos keluvchi qator hadi izlanayotgan miqdorning taqribiy qiymati deb olinadi. Odatda yaqinlashish sharti sifatida shart qabul qilinadi. Bu erda eps oldindan berilgan son. Qator hadlarini rekurrent formulalar yordamida ifodalash dasturlashni ancha engillashtiradi.
Matematikada ? sonini ?/4=1-1/2!+1/3!-1/4!+...+(-1)(i+1)*1/i!+... cheksiz qator yordamida hisoblash mumkinligi isbotlangan. Bu qatorni quyidagi rekurrent formulalar yordamida ifodalash mumkindir:
R1=1.0, S1=1.0,Ri=-ri-1*(1/i), Si=Si-1+Ri;
Bu masalani echishning while operatori yordamida tuzilgan dasturini kurib chiqamiz:

```
#include <iostream.h>
{  double eps;
  Cout<<("\n eps="); Cin>>("%f",&eps);
 int i=2;
 double r=1.0;
 double s=1.0;
  while(r>eps||r<-eps);
{ s+=r;
   r= - r*(1/i);
   i++;
}
Cout<<("pi=%f",s*4);
}
```

Shu masalaning do while operatori yordamida tuzilgan dasturi:

```
#include <iostream.h>
{  double eps;
```

```
  Cout<<(“\n eps=”); Cin>>(“%f”,&eps);
 int i=1;
 double r=1.0;
 double s=0.0;
  do
{ s+=r;
   r=-r*(1/i);
   i++;
}
while(r=>eps||r<=-eps);
Cout<<(“pi=%f”,s*4);
}
```

Shunga e'tibor berish kerakki tekshirish tsikl tanasi bajarilgandan sung amalga oshirilgani uchun kichik eki teng sharti quyilgandir.

Shu masalani for operatori yordamida tuzilgan dasturi:

```
#include <iostream.h>
{  double eps;
  Cout<<(“\n eps=”); Cin>>(“%f”,&eps);
  for(int i=1, double r=1.0,double s=1.0; r>eps||r<-eps;i++)
{ r=-r*(1/i);s+=r};
Cout<<(“pi=%f”,s*4);
}
```

Leksik analiz. Kiritilgan ifoda haqiqiy sonligini tekshiruvchi dastur:

```
       #include <iostream.h>
void Main()
{
int k=1; m=0;

while (c!=’\n’) {
if (c=’.’ && m=0) {m=1;continue};
if (c<’0’ || c>’9’) {k=0;break};
 }
if  (k) Cout<<“\n Hakikiy son”; else  Cout<<“\n Hakikiy son emas”;
}
```

Keyingi dasturimizda kiritilaetgan ifoda identifikator yoki yuqligi tekshiriladi:

```
      #include <iostream.h>
void Main()
{
int k=0;

while (c!=’\n’)
{if (k==-1) break;
 m=2;
if  (c>=’0’ && c<=’9’) m=0;
if  (c>=’a’ && c<=’Z’) m=1;
if (c==’_’) m=1;
}
switch(m)
{
case 0: if (k==0) k=-1;break;
```

```
case 1: k=1;beak;
default k=-1;break;
}
}

if  (k=-1) Cout<<(“\n Identifikator emas”); else  Cout<<(“\n Identifikatoras”);
}
```

22 – DARS. SATR, FUNKTSIYA, GRAFIKA.

Foydalanuvchi Funktsiyalari.

Funktsiyalarni ta'riflash va ularga murojaat kilish. Funktsiya ta'rifida funktsiya nomi, tipi va formal parametrlar ruyhati ko'rsatiladi. Formal parametrlar nomlaridan tashqari tiplari ham ko'rsatilishi shart. Formal parametrlar ro'yhati funktsiya signaturasi deb ham ataladi. Funktsiya ta'rifi umumiy kurinishi kuyidagichadir:

```
Funktsiya tipi funktsiya nomi(formal_parametrlar_ta'rifi)
```

Formal parametrlarga ta'rif berilganda ularninga boshlang'ich qiymatlari ham ko'rsatilishi mumkin. Funktsiya qaytaruvchi ifoda qiymati funktsiya tanasida return <ifoda> ; operatori orqali ko'rsatiladi.

Misol:

```
Float  min(float, float b)
{ if (a<b) return a;
       return b;
}
```

Funktsiyaga murojaat qilish quyidagicha amalga oshiriladi:

Funktsiya nomi (haqiqiy parametrlar ruyhati)

Haqiqiy parametr ifoda ham bo'lishi mumkin. Haqiqiy parametrlar qiymati hisoblanib mos formal parametrlar o'rnida ishlatiladi.

Misol uchun yuqoridagi funktsiyaga qo'yidagicha murojaat qilish mumkin:

```
Int x=5,y=6,z; z=min(x,y) eki int z=Min(5,6) eki int x=5; int z=min(x,6)
```

Funktsiya ta'rifida formal parametrlar initsializatsiya qilinishi, ya'ni boshlang'ich qiymatlar ko'rsatilishi mumkin. Funktsiyaga murojaat qilinganda biror haqiqiy parametr ko'rsatilmasa, uning urniga mos formal parametr ta'rifida ko'rsatilgan boshlang'ich qiymat ishlatiladi.

Misol uchun:

```
Float  min(float a=0.0, float b)
{ if (a<b) return a;
       return b;
}
```

Bu funktsiyaga yuqorida ko'rsatilgan murojaat usullaridan tashqari quyidagicha murojaat qilish mumkin:

```
Int y=6,z; z=min(,y) eki int z=Min(,6);
```

Agar funktsiya hech qanday qiymat qaytarmasa uning tipi void deb ko'rsatiladi.

Misol uchun:

```
Void print;
{ Cout<<("\n Salom!);
};
```

Bu funktsiyaga Print; shaklida murojjat qilish ekranga Salom! Yozilishiga olib keladi.

Qiymat qaytarmaydigan funktsiya formal parametrlarga ega bo'lishi mumkin:

```
Void Pint_Baho(Int baho);
{
Switch(baho)
{case 2:Cout<<("\n  emon");break;
case 3:Cout<<("\n  urta");break;
case 4:Cout<<("\n  yahshi");break;
case 5:Cout<<("\n  a'lo");break;
default: Cout<<("\n  baho notugri kiritilgan");
};
```

Bu funktsiyaga Print_Baho(5) shaklida murojaat qilish ekranga a'lo so'zi yozilishiga olib keladi.
Agar programmada funktsiya ta'rifi murojaatdan keyin berilsa, yoki funktsiya boshqa faylda joylashgan bo'lsa, murojjatdan oldin shu funktsiyaning prototipi joylashgan bulishi kerak. Prototip funktsiya nomi va formal parametrlar tiplaridan iborat bo'ladi. Formal parametrlar nomlarini berish shart emas.
Misol uchun y=min(a,b)+2*max(c,d) ifodani hisoblashni kuramaz:

```
#Include <iostream.h>
int max(int a,int b)
{if (a<b) return a;else return b};
void main()
{int a,b,c,d,y;
 int min(int ,int);
 Cin>>("\n %f%f%f%f",&a,&b,&c,&d);
y=min(a,b)+2*max(c,d);
Cout<<("\n %f",y);
};
int min(int a,int b)
{if (a<b) return b;else return a};
```

Funktsiyaga parametrlar uzatish. Funktsiyaga parametrlar qiymat buyicha uzatiladi va quyidagi bosqichlardan iborat bo'ladi:
1. Funktsiya bajarishga tayyorlanganda formal parametrlar uchun hotiradan joy ajratiladi, ya'ni formal parametrlar funktsiyalarning ichki parametrlariga aylantiriladi. Agar parametr tipi float bo'lsa double tipidagi ob'ektlar hosil buladi, char va shortint bulsa int tipidagi ob'ektlar yaratiladi.
2. Haqiqiy parametrlar sifatida ishlatilgan ifodalar qiymatlari hisoblanadi.
3. Haqiqiy parametrlar ifodalar qiymatlari formal parametrlar uchun ajratilgan hotira qismlariga yoziladi. Bu jarayonda float tipi double tipiga, char va shortint tiplari int tipiga keltiriladi.
4. Funktsiya tanasi ichki ob'ektlar – parametrlar yordamida bajariladi va qiymat chaqirilgan joyga qaytariladi.
5. Haqiqiy parametrlar qiymatlariga funktsiya hech qanday ta'sir o'tkazmaydi.
6. Funktsiyadan chiqishda formal parametrlar uchun ajratilgan hotira qismlari bo'shatiladi.

C ++ tilida chaqirilgan funktsiya chaqiruvchi funktsiyadagi o'zgaruvchi qiymatini uzgartira olmaydi. U faqat o'zining vaqtinchalik nushasini o'zgartirishi mumkin holos. Qiymat bo'yicha chaqirish qulaylik tug'diradi. Chunki funktsiyalarda kamroq o'zgaruvchilarni ishlatishga imkon beradi. Misol uchun shu hususiyatni aks ettiruvchi POWER funktsiyasi variantini keltiramiz:

```
power(x,n)  /* raise  x  n-th power; n > 0;
version 2 */
int x,n;
int p;
for (p = 1; n > 0; --n)
p = p * x;
return (p);
```

Argument N vaqtinchalik o'zgaruvchi sifatida ishlatiladi. Undan to qiymati 0 bo'lmaguncha bir ayriladi. N funktsiya ichida o'zgarishi funktsiyaga murojjat qilingan boshlang'ich qiymatiga ta'sir qilmaydi.

23 – DARS. REKURSIYA.

Rekursiv funktsiyalar. Rekursiv funktsiya deb o'ziga uzi murojjat qiluvchi funktsiyaga aytiladi. Misol uchun faktorialni hisoblash funktsiyasini keltiramiz:

```
Long fact(int k)
{if (k<0) return 0;
if (k==0) return 1;
return k*fact(k-1);
}
```

Manfiy argument uchun funktsiya 0 qiymat qaytaradi. Parametr 0 ga teng bo'lsa funktsiya 1 qiymat qaytaradi. Aks holda parametr qiymat birga kamaytirilgan holda funktsiyaning o'zi chaqiriladi va uzatilgan parametrga ko'paytiriladi. Funktsiyaning uz uzini chaqirish formal parametr qiymati 0 ga teng bo'lganda tuhtatiladi. Keyingi misolimizda ihtiyoriy haqiqiy sonning butun darajasini hisoblash rekursiv funktsiyasini keltiramiz.

```
Double expo(double a, int n)
{ if (n==0) return 1;
if (a==0.0) return 0;
if (n>0) return a*expo(a,n-1);
if(n<0) return expo(a,n+1)/a;
}
```

Misol uchun funktsiyaga expo(2.0,3) shaklda murojaat qilinganda rekursiv ravishda funktsiyaning ikkinchi parametri kamaygan holda murojjatlar hosil buladi: Expo(2.0,3),expo(2.0,2),expo(2.0,1),expo(2.0,0). Bu murojaatlarda quyidaga kupaytma hisoblanadi: 2.0*2.0*2.0*1 va kerakli natija hosil qilinadi.

Shuni kursatib o'tish kerakki bu funktsiyamizda noaniqlik mavjuddir ya'ni 0.0 ga teng sonning 0 chi darajasi 0 ga teng bo'ladi. Matematik nuqtai nazardan bo'lsa bu holda noaniqlik kelib chiqadi. Yuqoridagi sodda misollarda rekursiyasiz iterativ funktsiyalardan foydalanish maqsadga muvofiqdir.

Masalan darajani hisoblash funktsiyani quyidagicha tuzish mumkin:

```
Double expo(double a, int n)
{ if (n==0) return 1;
if (a==0.0) return 0;
int k=(n>0)?n:-n;
for(double s=1.0,int i=0;i<k;i++,s*=a);
if (n>0) return s else return 1/s;
}
```

Rekursiyaga misol sifatida sonni satr shaklida chiqarish masalasini ko'rib chiqamiz. Son raqamlari teskari tartibda hosil buladi. Birinchi usulda raqamlarni massivda saqlab so'ngra teskari tartibda chiqarishdir.

Rekursiv usulda funktsiya har bir chaqiriqda bosh raqamlardan nusha olsih uchun o'z o'ziga murojaat qiladi, so'ngra ohirgi rakamni bosib chiqaradi.

```
printd(n)   /* print n in decimal (recursive)*/
int n;
(
int i;
if (n < 0)
putchar('-');
n = -n;
```

```
if ((i = n/10) != 0)

printd(i);
putchar(n % 10 + '0');
)
```

PRINTD(123) chaqiriqda birinchi funktsiya PRINTD N = 123 qiymatga ega. U 12 qiymatni ikkinchi PRINTD ga uzatadi, boshqarish o'ziga qaytganda 3 ni chiqaradi.

24 – DARS. HOTIRA SINFLARI.

Ob'ektlarni lokallashtirish.

Blok deb funktsiya tanasi eki figurali qavslar ichiga olingan ta'riflar va operatorlar ketma ketlishgiga aytiladi.

Avtomatik hotira ob'ektlari faqat o'zi aniqlangan blok ichida mavjud bo'ladi. Blokdan chiqishda ob'ektlar uchun ajratilgan hotira qismi bo'shatiladi, ya'ni ob'ektlar yuqoladi. Shunday qilib avtomatik hotira har doim ichki hotiradir, ya'ni bu hotiraga o'zi aniqlangan blokda murojaat qilish mumkin. Avtomatik hotira ob'ektlari auto yoki register so'zlari yordamida ta'riflanadi. Agar mahsus ko'rsatilmagan bo'lsa o'zgaruvchi har doim avtomatik hotira turiga tegiishli deb hisoblanadi. Statik hotira ob'ektlari blokdan chiqilgandan so'ng ham mavjud bo'lib qolaveradi. Statik hotira ob'ektlari statik hizmatchi so'zi yordamida ta'riflanadi.

Misol:

```
#Include <iostream.h>
void autofunc(void)
{ int K=1;
 Cout<<(“\K=%d”,K);
K++;
Return;
}
void main()
{
int i;
for (i=0;i<5;i++)
autofunc();
}
```

Bu dastur bajarilishi natijasi:

K=1 K=1 K=1 K=1 K=1

Shu dasturning ikkinchi ko'rinishida K o'zgaruvchi statik o'zgaruvchi sifatida ta'riflanadi:

```
#Include <iostream.h>
void autofunc(void)
{ static int K=1;
 Cout<<(“\K=%d”,K);
K++;
Return;
}
void main()
{
int i;
for (i=0;i<5;i++)
autofunc();
}
```

Bu dastur bajarilishi natijasi:

K=1 K=2 K=3 K=4 K=5

Bu misolda K o'zgaruvchi faqat bir marta initsializatsiya qilinadi va uning qiymati navbatdagi murojaatgacha saqlanadi.

25 – DARS. GLOBAL OB'EKTLAR.

Global ob'ektlar deb dasturda hamma bloklar uchun umumiy bo'lgan ob'ektlarga aytiladi. Har bir blok ichida global ob'ektga murojjat qilish mumkindir.

Misol:

```
#include <iostream.h>
int N=5;
void func(void)
{
Cout<<(“\tN=%d”,N);
N--;
Return;
}
void main()
{
int i;
for (i=0;i<5;i++)
{func();
N+=2;
}
}
```

Dastur bajarilishi natijasi:

N=5 N=6 N=7 N=8 N=9

N o'zgaruvchisi main() va func() funktsiyalari tashqarisida aniqlangan va bu funktsiyalarga nisbatan globaldir. Har bir func() chaqirlganda uning qiymati 1 ga kamayadi, asosiy dasturda bo'lsa 2 taga oshadi. Natijada N qiymati 1 ga oshib boradi.

Endi dasturni o'zgartiramiz:

```
#include <iostream.h>
int N=5;
void func(void)
{
Cout<<(“\tN=%d”,N);
N--;
Return;
}
void main()
{
int N;
for (N=0;N<5;N++)
{func();
N+=2;
}
}
```

Dastur bajarilishi natijasi:

N=5 N=4 N=3 N=2 N=1

N uzgaruvchisi main() funktsiyasida avtomatik o'zgaruvchi sifatida ta'riflangan va u global N o'zgaruvchiga ta'sir qilmaydi. Ya'ni global N o'zgaruvchi main() funktsisida ko'rinmaydi.

Tashqi ob'ektlar.

Dastur bir necha matnli fayllarda joylashgan bo'lishi mumkin. Dastur o'z navbatida funktsiyalardan iboratdir. Hamma funktsiyalar tashqi hisoblanadi. Hatto har hil fayllarda

joylashgan funktsiyalar ham har hil nomlarga ega bo'lishi lozimdir. Funktsiyalardan tashqari dasturlarda tashqi ob'ektlar o'zgaruvchilar, ko'rsatkichlar, massivlar ishlatilishi mumkin.
Tashqi ob'ektlar hamma funktsiyalarda ham ko'rinmasligi mumkin. Agar ob'ekt fayl boshida ta'riflangan bo'lsa u shu fayldagi hamma funktsiyalarda ko'rinadi.
Agar tashqi ob'ektga shu ob'ekt ta'riflangan blokdan yuqorida eki boshqa faylda joylashgan blokdan murojjat qilinishi kerak bo'lsa, bu ob'ekt extern hizmatchi so'zi yordamida ta'riflanshi lozimdir. Shuni aytish lozimki extern hizmatchi so'zi yordamida ta'riflanganda initsializatsiya qlish yo chegaralarni ko'atish mumkin emas.

```
Extern double summa[];
Extern char D_phil [];
Extern long M;
```

Misol uchun VAL va SP o'zgaruvchilari bitta faylda, bu o'zgaruvchilarga murojaat qiluvchi PUSH, POP i CLEAR funktsiyalari boshqa faylda ta'riflangan bo'lsin. Bu holda bu fayllar orasidagi bog'liklikni ta'minlash uchun quyidagi ta'riflar lozimdir:
1 faylda:

```
int sp = 0;  /* stack pointer */
double val[maxval]; /* value stack */
```

2 faylda:

```
extern int sp;
extern double val[];
double push(f)  ...
double pop()   ...
clear()   ...
```

26 – DARS. DINAMIK HOTIRA.

Dinamik hotira bu dastur bajarilishi jarayonida ajratiladigan hotiradir. Dinamik hotira ajratilgandan so'ng to free() funktsiyasi tomonidan bo'shatilmaguncha saqlanadi. Agar dinamik hotira dastur bajarilishi tugaguncha bo'shatilmagan bo'lsa, avtomatik ravishda dastur tugaganda bo'shatiladi. Shunga qaramay ajratilgan hotirani dasturda mahsus bo'shatish dasturlashning sifatini oshiradi.

Dastur bajarilish davomida ajratilgan hotira qismiga murojaat imkoniyati shu qismga adreslovchi ko'rsatkichga bog'likdir. Shunday qilib, biror blokda ajratilayotgan dinamik hotira bilan ishlashning quyidagi uchta varianti mavjuddir:

- Ko'rsatkich avtomatik hotira turiga kiruvchi lokal ob'ekt. Bu holda ajratildgan hotiraga blokdan tashqarida murojaat qilib bo'lmaydi, shuning uchun blokdan chiqishda bu hotirani bo'shatish lozimdir.
- Ko'rsatkich avtomatik hotira turiga kiruvchi lokal statitk ob'ekt. Blokda bir marta ajratilgan dinamik hotiraga, blokka har bir qayta kirilganda ko'rsatkich orqali murojaat qilish mumkindir. Hotirani blokdan foydalanib bo'lgandan so'ng bo'shatish lozimdir.
- Ko'rsatkich blokka nisbatan global ob'ektdir. Dinamik hotiraga ko'rsatkich ko'rinuvchi hamma bloklardan murojaat qilish mumkin.Hotirani fakat undan foydalanib bo'lgandan so'ng b'oshatish lozimdir.

Ikkinchi variantga misol keltiramiz, bu misolda dinamik hotira ob'ekti ichki statik k'orsatkich bilan bog'likdir:

```
Include <stdio.h>
Include <alloc.h>
Void dynamo(void)
  {
    static char *uc=NULL;
     if (uc==NULL)
        {
           uc=(char*)malloc(1);
           *uc='A';
         }
printf("\t%c",*uc);
(*uc)++;
return;
};
void main()
{
   int I;
   for (i=0;i<5;i++)
    dynamo();
};
```

Dastur bajarilishi natijasi:

A B C D E

Bu dasturning kamchiligi ajratilgan hotira free() funktsiyasi yordamida bo'shatilmasligidir. Keyingi dasturda dinamik hotiraga ko'rsatkich global ob'ektdir:

```
Include <stdio.h>
Include <alloc.h>
char *uk=NULL;
void dynam1(void)
  {
    printf("\t%c",*uc);
```

```
    (*uk)++;
    return;
  };
void main()
{
  int I;
  uk=(char*)malloc(1);
        *uk='A';
  for (i=0;i<5;i++)
   {
   dynam1();
   (*uk)++;
   }
free(uk);
};
```

Dastur bajarilishi natijasi:

A C E G I

Dinamik ob'ekt asosiy dasturda yaratilib, u ko'rsatkich bilan bog'likdir. Dasturda boshlang'ich 'A' qiymatga ega bo'ladi. Ko'rsatkich global bo'lgani uchun dinamik ob'ektga main() va dynamic() funktsiyalarida murojaat qilish mumkin. Dinamik hotiraga ajratilgandan so'ng shu ob'ekt bilan bog'lik ko'rsatkich tashqi ob'ekt sifatida ta'riflangan ihtiyoriy blokda murojaat qilish mumkin.

27 – DARS. MASSIVLAR .

Bir ulchovli massivlar.

Massiv bu bir tipli nomerlangan ma'lumotlar jamlanmasidir. Massiv indeksli o'zgaruvchi tushunchasiga mos keladi. Massiv ta'riflanganda tipi, nomi va indekslar chegarasi ko'rsatiladi. Misol uchun long int a[5]; char w[200];double f[4][5][7]; char[7][200]. Massiv indekslar har doim 0 dan boshlanadi. C ++ tili standarti bo'yicha indekslar soni 31 tagacha bo'lishi mumkin, lekin amalda bir o'lchovli va ikki o'lchovli massivlar qo'llaniladi. Bir ulchovli massivlarga matematikada vektor tushunchasi mos keladi. Massivning int z[3] shakldagi ta'rifi, int tipiga tegishli z[0],z[1],z[2] elementlardan iborat massivni aniqlaydi.

Massivlar ta'riflanganda initsializatsiya qilinishi, ya'ni boshlang'ich qiymatlarlari ko'rsatilishi mumkin. Misol uchun:

```
float C[]={1,-1,2,10,-12.5};
```

Bu misolda massiv chegarasi avtomatik aniqlanadi. Agar massiv initsializatsiya qilinganda elementlar chegarasi ko'rsatilgan bo'lsa , ruyhatdagi elementlar soni bu chegaradan kam bo'lishi mumkin, lekin ortiq bo'lishi mumkin emas. Misol uchun int A[5]={2,-2}. Bu holda a[0] va a[1] qiymatlari aniqlangan bo'lib, mos holda 2 va –2 ga teng.

Massivda musbat elemenlar soni va summasini hisoblash.

```
# include <iostream.h>;
# include <conio.h>;
Main() {
Int  x[]={-1;2;5;-4;8;9};
Clrscr();
For (int s=0,int k=0, int I=0; I<6; I++) {
If (x[I]<=0) continue;
k++;s++;
};
Cout<<("%d",k);
Cout<<("%d",k);
getch();
};
```

Massivning eng katta, eng kichik elementi va o'rta qiymatini aniqlash:

```
#include <iostream.h>
Void main()
{
Int I,j,n;
Float a,b,d,x[100];
While(1)
{
Cout<<("\n n="); Cin>>("%i",&n);
If ( n>0 && n < = 100 ) break;
Cout<<("\n Hato  0<n<101 bulishi kerak");
}
Cout<<("\n elementlar kiymatlarini kiriting:\n");
For (i=0;i<n;i++)
{ Cout<<("x[%i]=",i);Cin>>("%f",&x[i]);}
max=x[0];min=x[0];
For (s=0,i=0;i<n;i++)
```

```
{ s++;
  If (max<x[i]) max=x[i];
  If (min>x[i]) min=x[i];
};
s/=n;
Cout<<(“\n max=%f”,max);
Cout<<(“\n min=%f”,min);
Cout<<(“\n urta kiymat=%f”,s);
}
```

28 – DARS. JADVALLAR.

Ikki ulchovli massivlar matematikada matritsa yoki jadval tushunchasiga mos keladi. Jadvallarning initsializatsiya qilish qoidasi, ikki o'lchovli massivning elementlari massivlardan iborat bo'lgan bir o'lchovli massiv ta'rifiga asoslangandir. Misol uchun ikki qator va uch ustundan iborat bo'lgan haqiqiy tipga tegishli d massiv boshlang'ich qiymatlari qo'yidagicha ko'rsatilishi mumkin:

```
float d[2][3]={(1,-2.5,10),(-5.3,2,14)};
```

Bu yozuv quyidagi qiymat berish operatorlariga mosdir:

```
d[0][0]=1;d[0][1]=-2.5;d[0][2]=10;d[1][0]=-5.3;d[1][1]=2;d[1][2]=14;
```

Bu qiymatlarni bitta ro'yhat bilan hosil qilish mumkin:

```
float d[2][3]={1,-2.5,10,-5.3,2,14};
```

Initsializatsiya yordamida boshlang'ich qiymatlar aniqlanganda massivning hamma elementlariga qiymat berish shart emas.

Misol uchun: `int x[3][3]={(1,-2,3),(1,2),(-4)}`.

Bu yozuv qo'yidagi qiymat berish operatorlariga mosdir:

```
x[0][0]=1;x[0][1]=-2;x[0][2]=3;x[1][0]=-1;x[1][1]=2;x[2][0]=-4;
```

Initsializatsiya yordamida boshlang'ich qiymatlar aniqlanganda massivning birinchi indeksi chegarasi ko'rsatilishi shart emas, lekin qolgan indekslar chegaralari ko'rsatilishi shart.

Misol uchun:

```
Double x[][2]={(1.1,1.5),(-1.6,2.5),(3,-4)}
```

Bu misolda avtomatik ravishda qatorlar soni uchga teng deb olinadi.

Qo'yidagi ko'radigan misolimizda jadval kiritilib har bir qatorning maksimal elementi aniqlanadi va bu elementlar orasida eng kichigi aniqlanadi:

```
#include <iostream.h>
void main()
{ double a[4,3]; double s,max=0.0,min=0.0;
int i,j;
for(i=0;i<4;i++) {
for(j=0;j<3;j++)
{ Cout<<(" a[%d][%d]=",i,j);Cin>>("%f",s);a[i,j]=s;
if (max<s) max=s;
};
Cout<<("\n");
if (max<min) min=max;
}
Cout<<("\n min=%f",min);
}
```

Simvolli massivlar.

C ++ tilida satrlar simvolli massivlar sifatida ta'riflanadi. Simvolli massivlar qo'yidagicha tasvirlanishi mumkin: `Char pas[10];`

Simvolli massivlar qo'yidagicha initsializatsiya qilinadi:

`Char capital[]="TASHKENT";` Bu holda avtomatik ravishda massiv elementlari soni aniqlanadi va massiv ohiriga satr ko'chirish '\n' simvoli qo'shiladi.

Yuqoridagi initsializatsiyani qo'yidagicha amalga oshirish mumkin:

```
Char capital[]={‘T’,’A’,’S’,’H’,’K’,’E’,’N’,’T’,’\n’};
```
Bu holda so’z ohirida ‘\n’ simvoli aniq ko’rsatilishi shart.
Misol uchun palindrom masalasini ko’rib chiqamiz. Palindrom deb oldidan ham ohiridan ham bir hil o’qiladigan so’zlarga aytiladi. Misol uchun non. Dasturda kiritilgan so’z palindrom ekanligi aniqlanadi:
```
#include <iostream.h>
void main()
{
  gets(a);
for( int j=0, a[j]!=’\0’;j++);
I=0;
while(I<j) if (a[I++]!=a[j--]) break;
if ((j-I)>1) Cout<<(“Palindrom emas”) else Cout<<(“Palindrom”);
```
Keyingi misolimizda kiritilgan so’zdan berilgan harf olib tashlash dasturi berilgan:
```
#include <iostream.h>
void main()
{
char s[];
int c;
gets(a);
int i, j;
for ( i = j = 0; s[i] != '\0'; i++)
if ( s[i] != c )
s[j++] = s[i];
s[j] = '\0';
puts(s);
}
```
Har gal 's' dan farqli simvol uchraganda , u J pozitsiyaga yoziladi va faqat shundan so’ng J qiymati 1 ga oshadi. Bu qo’yidagi yozuvga ekvivalent:
```
if ( s[i] != c )
s[j] = s[i];
j++;
```

29 – DARS. SO’ZLAR MASSIVLARI.

C ++ tilida so’zlar massivlari ikki o’lchovli simvolli massivlar sifatida ta'riflanadi. Misol uchun:

```
Char Name[4][5].
```

Bu ta'rif yordamida har biri 5 ta harfdan iborat bo’lgan 4 ta so’zli massiv kiritiladi. So’zlar massivlari qo’yidagicha initsializatsiya qilinishi mumkin:

```
 Char Name[3][8]={“Anvar”,”Mirkomil”,”Yusuf”}.
```

Bu ta'rifda har bir so’z uchun hotiradan 8 bayt joy ajratiladi va har bir so’z ohiriga ‘\0’ belgisi qo’yiladi.

So’zlar massivlari initsializatsiya qilinganda so’zlar soni ko’rsatilmasligi mumkin. Bu holda so’zlar soni avtomatik aniqlanadi:

```
Char comp[][9]={“komp'yuter”,”printer”,”kartridj”}.
```

Quyidagi dasturda berilgan harf bilan boshlanuvchi so’zlar ruyhati bosib chiqariladi:

```
#include <iostream.h>
void main()
{ char a[10][10];
    char c;
for (int i=0;i<10;i++) gets(a[i]);
c=getchar();
for (i=0;i<10;i++) if (a[i][0]==c) puts(a[i]);
}
```

Qo’yidagi dasturda fan nomi, talabalar ruyhati va ularning baholari kiritiladi. Dastur bajarilganda ikki olgan talabalar ruyhati bosib chiqariladi:

```
#include <iostream.h>
void main()
{ char a[10][10];
   char s[10];
   int k[10];
gets(s);
for (int i=0;i<10;i++) gets(a[i]);
for (i=0;i<10;i++) {Cin>>(“%d”,k[i]};
for (int i=0;i<10;i++) if (k[i]==2) puts(a[i]);
}
```

30 – DARS. KO'RSATKICHLAR MASSIVLARI.

Ko'rsatkichlar massivlari qo'yidagicha ta'riflanadi

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

Misol uchun int *pt[6] ta'rif int tipidagi ob'ektlarga olti elementli massivni kiritadi.

Ko'rsatkichlar massivlari satrlar massivlarini tasvirlash uchun qulaydir. Misol uchun familiyalar ruyhatini kiritish uchun ikki ulchovli massivdan foydalani kerak. char fam[][20]={"Olimov","Rahimov","Ergashev"}

Hotirada 60 elementdan iborat bo'ladi, chunki har bir familiya gacha 0 lar bilan to'ldiriladi. Ko'rsatkichlar massivi yordamida bu massivni qo'yidagicha ta'riflash mumkin.

Char *pf[]= {"Olimov","Rahimov","Ergashev"}.

Bu holda ruyhat hotirada 23 elementdan iborat bo'ladi, chunki har bir familiya ohiriga 0 belgisi qo'yiladi

Ko'rsatkichlar massivlari murakkab elemenlarni sodda usulda tartiblashga imkon beradi. Quyidagi misolda matritsa satrlari birinchi elementlari o'sishi tartibida chiqariladi. Bu misolda yordamchi ko'rsatkichlar massivi yaratilib shu massiv tartiblanadi va massiv asosida matritsa elementlari chiqariladi.

```
# include <iostream.h>
void main()
{int n=2;
 int m=3;
array[][3]={(1,3,5),(3,1,4),(5,7,1)};
int *pa[n];
for (I=0;I<n;I++) pa[I]=(int *)&a[I];

for (I=o;I<n-1;I++)
{for (int k=I=1;k<n;k++)
if a[I][1]>a[k][1]
{ int *pp=pa[I];
pa[I]=pa[k];pa[k]=pp;};

for (I=0;I<n;I++)
{Cout<<("\n%I",I+1);
for (int j=0;j<magistr;j++)
Cout<<("%I",pa[I][j]);}
};
```

Ko'rsatkichlar massivlari funktsiyalarda matritsalar qiymatlarini o'zgartirish uchun mumkin. Qo'yidagi misolda matritsani transponirlash funktsiyasi ishlatiladi.

```
Void trans(int n,double *p[]);
{ double x;
for (int I=0;I<n-1;I++)
for (int j=I+1;j<n;j++)
{x=p[I][j];
p[I][j]=p[j][I];
p[j][I]=x;
}
};
void main()
{double a[3,3]={11,12,13,21,22,23,31,32,33};
double ptr={(double*)&a[0], (double*)&a[1], (double*)&a[2]};
```

```
int n=3;
trans(n,ptr);
for (int I=0;I<n;I++)
{Cout<<(“\n  %i”,i+1);
for (int j=0;j<n;j++)
Cout<<(“\n  %f”,a[I][j]);
};
};
```

31 – DARS. FUNKTSIYALAR VA MASSIVLAR.

Funktsiyalar va sonli massivlar.

Funktsiyalarda bir o'lchovli sonli massivlar argument sifatida ishlatilganda ularning chegarasini ko'rsatish shart emas. Misol tariqasida n o'lchovli vektorlar bilan bog'lik funktsiyalarni ta'riflashni ko'rib chiqamiz.

Vektorning uzunligini aniqlash funktsiyasi:

```
float mod_vec(int n,float x[])
{ float a=0;
for (int I=0;I<n;I++)
 a+=x[I]*x[I];
return sqrt(double(a));
}
```

Ikki vektorning skalyar kupaytmasi funktsiyasi:

```
float scalar(int n,float x[],float y[])
{ float a=0;
for (int I=0;I<n;I++)
 a+=x[I]*y[I];
return a;
}
```

Qo'yidagi dasturda ikki vektor orasidagi burchak kosinusini hisoblash funktsiyasi kiritiladi va bu funktsiya yordamida berilgan vektorlar orasidagi burchak kosinusi hisoblanadi:

```
#include <iostream.h>
#include <math.h>
float cosinus(int n,float x[],float y[])
{ float a=0,b=0,c=0;
for (int I=0;I<n;I++)
{ a+=x[I]*y[I];
 b+=x[I]*x[I];
 c+=y[I]*y[I];
}
return a/sqrt(double(b*c));
}
void main()
{float e[]={1,2,3};
float g[]={-1,7,4};
Cout<<("\n%I",cosinus(3,e,g));
}
```

Funktsiyalarda bir o'lchovli massivlar qaytariluvchi qiymatlar sifatida ham kelishi mumkin. Misol uchun ikki vektor summasini hisoblovchi funktsiya protsedurani ko'ramiz:

```
Void sum_vec(int n, float a,float b, float c)
{
for(int i=0;i<n;i++,c[i]=a[i]+b[i]);
}
```

Bu funktsiyaga qo'yidagicha murojaat qilish mumkin:

```
Float a[]={1,-1.5,-2},b[]={-5.2,1.3,-4},c[3]; sum_vec(3,a,b,c);
```

Massiv qiymat qaytaruvchi funktsiya ta'rifini:

```
Float *sum_vec(int n,float a,float b)
{ float d[n];
   for(int i=0;i<n;i++,d[i]=a[i]+b[i]);
```

```
return d;
}
```
Bu funktsiyaga qo'yidagicha murojaat qilish mumkin:
```
Float a[]={1,-1.5,-2},b[]={-5.2,1.3,-4};float c[]=sum_vec(3,a,b);
```
Massivlarni tartiblash.
Keyingi misolimizda massivlarni qiymatlari o'sish tartibida joylashtirish funktsiyasi berilgan:
```
Void function sort(int n, double a[])
{ int i,j; double c;
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
if (a[i]>a[j]) {c=a[i];a[i]=a[j];a[j]=c;}
```
Bu usulda har bir element a[0] dan boshlab keyingi hamma elementlar bilan solishtiriladi. Biror element ko'rilayotgan a[i] dan kichik bo'lsa bu elementlar o'rin almashtiriladi. Ohir natijada a[i] urniga i dan n gacha elementlar ichida eng kichigi qo'yiladi.
Qo'yidagi funktsiya butun sonlar massivini Shell usuli asosida tartiblaydi. Bu usulda oldin bir biridan uzoqda joylashgan elementlar solishtiriladi. Elementlar orasidagi interval birgacha kamayib boradi.
```
shell(v, n)  /* sort v[0]...v[n-1]
into increasing order */
int v[], n;
int gap, i, j, temp;
for (gap = n/2; gap > 0; gap /= 2)
for (i = gap; i < n; i++)
for (j=i-gap; j>=0 && v[j]>v[j+gap]; j-=gap)
temp = v[j];
v[j] = v[j+gap];
v[j+gap] = temp;
```
Bu dasturda uchta tsikl ishlatilgan. Eng tashki tsikl elementlar orasidagi intervalni N/2 dan nol'gacha ikki martadan kamaytirib boradi. O'rta tsikl interval bilan ajratilgan elementlarni solishtiradi; eng ichki tsikl tartiblanmagan elementlar o'rnini almashtiradi. Interval ohiri birga teng bo'lgani uchun hamma elementlar tug'ri tartiblanadi.
Keyingi misolimizda berilgan x qiymat tartiblangan massivda majudmi yoki yukligini aniqlovchi funktsiyani ko'rib chiqamiz. Bu funktsiya ikkiga bo'lish usulidan foydalanadi. Massiv elementlari o'sish tartibida joylashgan bo'lishi kerak. Funktsiya agar x qiymat V massivda mavjud bo'lsa shu qiymat nomerini mavjud bo'lmasa –1 ni qaytaradi.
```
binary(x, v, n) /* find x in v[0]...v[n-1] */
int x, v[], n;
int low, high, mid;
low = 0;
high = n - 1;
while (low <= high)
mid = (low + high) / 2;
if (x < v[mid])
high = mid - 1;
else if (x > v[mid])
low = mid + 1;
else /* found match */ return(mid);
return(-1);
```

32 – DARS. FUNKTSIYALAR VA JADVALLAR

Jadvallar funktsiyalar argumentlari sifatida kelganda jadvallarning birinchi parametridan boshqa parametrining chegaralari ko'rsatilishi shartdir. Misol tariqasida uch o'lchovli kvadrat matritsani uch o'lchovli vektorga ko'paytirish funktsiyasini k'orib chiqamiz:

```
Void Umn_vec( float a[3][3],float b[3], float c[3])
{
for(I=0;i<3;++) {
c[I]=0;
for(j=0;j<3;j++)
c[I]+=a[I,j]*b[j];
  };
}
```

Har hil chegarali jadvallar bilan funktsiyalardan foydalanishning bir yuli bu oldindan kiritiluvchi konstantalardan foydalanishdir. Lekin asosiy yuli ko'rsatkichlar massivlaridan foydalanish. Matritsani vektorga ko'paytirish funktsiyasi ko'rsatkichlar massivlari yordamida qo'yidagicha yoziladi:

```
  Void Umn_vec( int n,float* a[],float b[], float c[])
{ int I,j;
for(I=0;i<n;++) {
c[I]=0;
for(j=0;j<n;j++)
c[I]+=a[I,j]*b[j];
  };
}
```

Matritsani matritsaga ko'paytirish funktsiyasi esa qo'yidagicha yoziladi:

```
  Void Umn_matr( float* a[],float* b[], float* c[])
{
int I,j,k;
for(I=0;i<n;++)
for(j=0;j<n;j++)
{c[I,j]=0;
for(k=0;k<n;k++)
c[I,j]+=a[I,k]*b[j,k];
  };
}
```

33 – DARS. FUNKTSIYALAR VA SIMVOLLI MASSIVLAR.

Funktsiyalarda satrli massivlar ishlatilganda ularning chegarasini ko'rsatish shart emas. Satrlarning uzunligini hisoblash len funktsiyasii qo'yidagicha ta'riflash mumkin:

```
Int len(char c[])
{ int m=0;
for(m=0;c[m]!='0';m++);
return m;
};
```

Shu funktsiyadan foydalanilgan dasturni keltiramiz:

```
Include #<iostream.h>
Int len(char c[])
{ int m=0;
while(c[m++]);
return m-1
}
void main()
{char e[]="Pro Tempore!";
Cout<<("\n%I",len(E));
};
```

PASKAL' tilida copy funktsiyasi mavjud bo'lib, berilgan satrning, berilgan pozitsiyasidan boshlab berilgan sondagi simvollarini ajratib olishga imkon beradi. Shu funktsiyani C++ tilida qo'yidagicha ta'riflash mumkin:

```
  Void copy(char a[], char b[], int k, int m)
{ int l;
for(int n=0; a[n]!='\0';n++);
if (k>m) {b[0]='\0';break};
if (k+m>n)  l=n-k else l=m;
for(int i=0;i<l;b[i++]=a[k+i]); b[i]='\0';
}
```

Berilgan satrni teskariga aylantiruvchi funktsiya:

```
reverse(char s[]s)    /* reverse string s in place */
int c, i, j;
for(i = 0, j = strlen(s) - 1; i < j; i++, j--)
c = s[i];
s[i] = s[j];
s[j] = c;
```

Keyingi misolimizda T qatorni S qator ohiriga o'lovchi STRCAT(S, T) funktsiyasini ko'rib chiqamiz:

```
strcat(s,t)      /* concatenate t to end of s */
char s[], t[];  /* s must be big enough */
int i, j;
i = j = 0;
while (s[i] != '\0') / *find end of s */
i++;
while((s[i++] = t[j++]) != '\0') /*copy t*/
```

34 – DARS. FUNKTSIYALAR VA SATRLI MASSIVLAR.

Satrli massivlar funktsiya argumenti sifatida ishlatilganda satrlarning umumiy uzunligi aniq ko'rsatilishi shartdir.
Misol tariqasida ihtiyoriy sondagi satrlar massivini alfavit bo'yicha tartiblash funktsiyasidan foydalanilgan dasturni ko'rib chiqamiz :

```
Include <iostream.h>
Define m 10
Void sort(int n, char a[][m]);
{char c[m];int I,l;
for (i=0;i<n;i++)
for (j=i+1;j<m;j++)
if  (a[i][0]<a[j][0] )
for (l=0;l<m;l++) {c=a[i][l];a[i][l]=a[j][l];a[j][l]=c;};
}
Void main()
{
  char aa[][m]={“Alimov”,”Dadashev”,”Boboev”};
    por(3,aa);
for(int i=0;i<3;i++) Cout<<(“\n %s”,aa[i]);
}
```

Bu misolda funktsiya spetsifikatsiyasida ko'rsatish shart bo'lgan satrlar massivining ikkinchi parametrini m konstanta orqali kiritdik.
Funktsiyada bu shartdan kutilishning eng umumiy yuli ko'rsatkichlar massividan foydalanishdir:

```
Include <iostream.h>
Void sort(int n, char* a[]);
{char* c;int I;
for (i=0;i<n;i++)
for (j=i+1;j<m;j++)
if  (a[i][0]<a[j][0] )
{c=a[i];a[i]=a[j];a[j]=c;};
}
Void main()
{ char* pa[];
  char aa[][3]={“Alimov”,”Dadashev”,”Boboev”};
for(int i=0;i<3;i++) pa[i]=&aa[i];
    sort(3,aa);
for(int i=0;i<3;i++) Cout<<(“\n %s”,aa[i]);
}
```

Bu misolda shunga e'tibor berish kerakki satrli massivning tartiblanmagan e'lementlarini almashtirish uchun qo'shimcha tsikl kiritilgan edi. Ko'rsatkichlar massivining elementlari adreslardan iborat bo'lgani uchun, qo'shimcha tsiklga hojat qolmadi. Ko'rsatkichlar massivini initsializatsiya qilish. Nchi oy nomidan iborat bo'lgan simvolli qatorga ko'rsatkich qaytaruvchi MONTH_NAME(N) funktsiyasini ko'rib chiqamiz. Bu funktsiyada ichki statik massiv ishlatilgan bo'lib, funktsiyana murojaat kerakli qatorga ko'rsatkich qayta oladi.

```
char *month_name(n) /* return name of n-th month */
int n;
\(
```

```
static char *name[] = \(
"illegal month",
"january",
"february",
"march",
"april",
"may",
"jun",
"july",
"august",
"september",
"october",
"november",
"december"
\);
return ((n < 1 \!\! n > 12) ? name[0] : name[n]);
\)
```

35 – DARS. E'LON FAYLLARI.

Standart kutubhona ichidagi funksiyalarni ishlatish uchun ularning prototiplari joylashgan e'lon fayllarini include preprosessor bo'yrug'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 "my_file.h"

36 – DARS. KO'RSATKICHLAR VA MASSIVLAR

Ko'rsatkichlar qiymati yuqorida ko'rsatilganidek bu biror ob'ekt adresidir. Ko'rsatkichlar ta'riflanganda ularning tiplari ko'rsatilishi shart. K'orsatkichlarni initsializatsiya qilish ya'ni boshlang'ich qiymatlarini kiritish mumkin. Ko'rsatkichlarga boshlangich mahsus NULL kiymati berilsa bunday kursatkich bush kursatkich deb ataladi.

Misol uchun:

```
Long double ld=0.0;
Long double *ldtr=&ld;
Int *pr;
 char *alfa;
unsigned longint *ul=NULL;
```

Mahsus void tipidagi ko'rsatkichlar ajdodiy ko'rsatkichlar deb atalib har hil tipdagi ob'ektlar bilan bog'lanish uchun ishlatiladi.

Misol uchun:

```
Int I=77;
Float Euler=2.18282;
Void *vp;
Vp=&I;
Cout<<("%d",*(int*)vp);
Vp=&euler;
Cout<<("%d",*(float*)vp);
```

Qo'yidagi operatorlar ketma ketligi hatolikka olib keladi:

```
Void *vp;int *ip; ip=vp;
```

Bu hatolik sababi bitta ob'ektga har hil tipdagi ko'rsatkichlar bilan murojaat qilish mumkin emas. Ko'rsatkichlar ustida qo'yidagi amallarni bajarish mumkin:

* operatsiyasi o'zgaruvchi qiymatini adres deb qarab shu adresda joylashgan qiymatni ajratadi.

& operatsiyasi o'zgaruvchi joylashgan adresni aniqlaydi.

= Qiymat berish
+ Binar plyus
- Binar minus
++ Oshirish
-- Kamaytirish
Nisbat amallari;

Adresni aniqash amaliga misol:

```
Unsigned int *up1=NULL, *up2;
Up2=(unsigned int*)&up1;
```

Qiymat berish operatorida tiplarni o'zgartirishni ko'rsatmaslik ya'ni up2=&up1 hatolikka olib keladi.

Ayirish amali ikki ko'rsatkich yoki ko'rsatkich va butun son ustida amalga oshirilishi mumkin. Ikki ko'rsatkich ayirmasi p1-p2=((long)p1-(long)p2)/sizeof(<tip>) formulasi orqali aniqlanadi. Ikki yonma-yon ko'rsatkichlar ayirmasi absolyut qiymati birga teng.

1- misol

```
char a='d',b='2';
char pa=&a,pb=&b;
U holda BC++ da
Pa-pb =1;
(int)pa-(int)pb=1;
```

pb-pa=-1 bo'ladi, chunki BC++da oldin ta'riflangan o'zgaruvchilar adreslari keyin ta'riflangan o'zgaruvchilar adreslaridan kattaroq bo'ladi.

2- misol

```
long a=12, b=3;
long pa=&a,pb=&b;
```

U holda BC++ da

```
Pa-pb =1;
(int)pa-(int)pb=4;
```

Ko'rsatkichdan k butun son ayrilganda qiymati k*sizeof(<tip>) ga kamayadi.
Qo'shish amalini faqat ko'rsatkich va songa qo'llash mumkin.
Misol:

```
Float z=0.0,pi=3.14,e=2.7;
Float *pr=&e;
```

U holda

```
*pr=2.7
*(pr+1)=3.14
*(pr+2)=0.0
```

++ va – amallari ham shu kabi bajariladi.
Massivlar nomi konstanta ko'rsatkichdir. Shuning uchun ham int z[4] massiv elementiga *(z+4) shaklda murojaat qilish mumkin.
Massivlar bilan ishlanganda qavslarsiz ishlash mumkin.
Qo'yidagi misolda har bir harf alohida bosib chiqariladi:

```
Void main()
{ char x[][="DIXI";
int=0;
while (*(x+I)!='\0')
Cout<<("/n%c",*(x=I++));
}
```

Kompilyator x[I] qiymatni hisoblaganda (x+i)*sizeof(<>) formula bo'yicha adresni hisoblab shu adresdagi qiymatni oladi. Agar massiv <tip> x[n,k,m] shaklda ta'riflangan bo'lsa, x[I,j,l] qiymatni hisoblanganda (x+k*j+l) *sizeof(<>) formula bo'yicha adresni hisoblab shu adresdagi qiymatni oladi.

Kursatkichlar va simvolli massivlar.

C ++ tili mualliflari massivlar o'rnida funktsiyalarda ko'rsatkichlardan foydalanishni maslahat beradilar. Shuni ta'kidlab o'tish kerakki massivlar nomlari funktsiya tanasida konstanta deb hisoblanmaydi, Shuning uchun bu ko'rsatkichga ++ va – amallarini qo'llash mumkindir. Misol tariqasida so'zlar bilan ko'rilgan funktsiyalarni faqat ko'rsatkichlardan foydalanilgan variantlarini ko'rib chiqamiz.
Satr uzunligini hisoblash:

```
Int len (char *s)
{ int m;
  for(m=0; *(s+m)!='\0';m++)
return m;
}
```

Keyingi variantda s kursatkichga ++ amali qo'llaniladi:

```
Int len (char *s)
{ int m;
  for(m=0; *s++!='\0';m++)
```

```
return m;
}
```
Bir satrdan ikkinchisiga nusha olish:
```
Void copy(char *c1, char *c2)
        {
          I=0;
     While(*(c1+i)=*(c2+I))1=’0’)
I++;
}
```
Nol'dan katta qiymat while operatori shartida rost hisoblangani uchun bu dasturni ++ amalini qo’llagan holda qo’yidagicha qisqarok yozish mumkin:
```
Void copy(char *c1, char *c2)
        {
          While(*c1++=*c2++);
}
```

Satrlarni ulash(konkatenatsiya funktsiyasi)
```
Void concat(char *c1,char *c2)
Int I;
For(m=0;*(c1+I)!=’\0’; m ++)
While((*(c1+m+I)=*(c2+I))
I++;
}
```
Bu funktsiyaning qisqa varianti:
```
Void concat(char *c1,char *c2)
     While(*c1++);
While(*c1++=*c2++)
}
```
Ikki satrni o’zaro solitirish:
Bu misolimizda ko’rsatkichlardan har hil foydalanish usullari ko’rsatiladi:
```
Int row(char c1[], char c2[])
{ int I,m1,m2;
  for(m1=0; *(c1+m1)=’\0’;m1++);
  for(m2=0;*(c2+m2)=’\0’,m2++)
  if (m1!=m2) return -1;
  for(I=0;I<m1;I++)
  if (*c1++  !=*c2++) return (I+1);
  return 0;
```
Bu funktsiya qo’yidagi qiymatlarni qaytaradi:
Agar satrlar uzunligi har hil bulsa 0;
Agar hamma simvollar mos kelsa 0;
Agar simvollar mos kelmasa birinchi mos kelmagan simvol nomeri.

37 – DARS. 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.

38 – DARS. 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

39 – DARS. 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.

40 – DARS. 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

41 – DARS.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. Etkanimizdek, 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
...
```

42 - DARS. FUNKSIYALAR VA UNING SHABLONLARI.

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
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.

DARS. 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.

43 - DARS. 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
```

44 - DARS. 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.

45 - DARS. 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 yordamida o'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 qilmàyapti. Ikkinchi holda esa o'zgaruchining qiymatini
o'zgartirsa bo'ladi, lekin pointer ko'rsatayî tgan 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 pointeri-
miz 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
```

46 - DARS. PREPROTSESSOR VOSITALARI.

Fayllardan matnlar qo'shish.
Fayldan matn kushish uchun uch shaklga ega bo'lgan # include operatori qo'llaniladi:
include <fayl nomi>
include "fayl nomi"
include makros nomi
Makros nomi #define direktivasi orqali kiritilgan preprotsessor identifikatori yoki makros bo'lishi mumkin. Agar birinchi shakl kullanilsa preprotsessor kushilaetgan faylni standart bibliotekalardan izlaydi. Agar ikkinchi shakl kullanilsa preprotsessor foydalanuvchining joriy katalogini ko'rib chiqadi va bu katalogda fayl topilmasa standart sistemali kataloglarga murojaat qiladi. C ++ standarti bo'yicha .h suffiksi bibliotekaga tegishli funktsiyalarning prototiplari hamda , tiplar va konstantalar ta'rifi joylashgan fayllarni ko'rsatadi. Bunday fayllarni sarlovhali fayllar deb ataladi. Kompilyator bibliotekalari bilan ishlashga muljallangan sarlovhali fayllar ro'yhati til standartida ko'rsatilgan bo'lib bu fayllar nomlari tilning hizmatchi suzlari hisoblanadi. Qo'yida shu standart fayllar nomlari keltirilgan:
Assert.h – programma diagnostikasi .
Type.h – simvollarni uzgartirish va tekshirish.
Erruo.h – hatolarni tekshirish.
Float.h – haqiqiy sonlar bilan ishlash.
Limit1.h – butun sonlarning chegaralari.
Locate.h – milliy muhitga moslash.
Match.h – matematik hisoblashlar.
Setjump.h – nolokal utishlar imkoniyatlari.
Sigual.h – gayrioddiy holatlar bilan ishlash.
Stdarg.h – o'zgaruvchi sonli parametrlarni qo'llash.
Stddef.h – qo'shimcha ta'riflar.
Iostream.h – kiritish-chikarish vositalari.
Stdlib.h – hotira bilan ishlash.
String,h – simvolli katorlar bilan ishlash.
Time.h – sana va vaqtni aniqlash.

Turbo C va Borland C++ kompilyatorlarida grafik biblioteka bilan boglanish uchun graphic.h – sarlavhali fayl kullaniladi.

Agar programmada bir necha funktsiyalardan foydalanilsa ,funktsiyalar ta'rifi ,tanasi bilan birga alohida fayllarda saqlash qulaydir. Hamma funktsiyalar tanasiga va main() funktsiyasi tanasiga chaqirilayotgan funktsiyalar prototiplari joylashtirilsa, programma tanasida funktsiyalarni ihtiyoriy joylashtirish mumkin. Bu holda programma fakat protsessor komandalaridan ham iborat bulishi mumkin.

47 - DARS. SHARTLI VA YORDAMCHI DIREKTIVALAR.

Shartli direktiva qo'yidagi ko'rinishga egadir:

```
#if  butun sonli ifoda.
tekst_1
#else
tekst_2
#endif
```

#else tekst_2 qismi ishlatilishi shart emas.

Direktiva bajarilganda #if dan so'ng yozilgan butun sonli ifoda qiymati hisoblanadi. Agar bu qiymat 0 dan katta bo'lsa tekst_1 kompilyatsiya qilinayotgan matnga qo'shiladi, aksincha tekst_2 qo'shiladi. Agar #else direktivasi va tekst_2 mavjud bo'lmasa bu direktiva o'tkazib yuboriladi.

#ifdef identifikator

direktivasida #define direktivasi yordamida identifikator aniqlanganligi tekshiriladi. Agar identifikator aniqlangan bo'lsa tekst_1 bajariladi.

#ifndef identifikator

direktivasida aksincha shart rost hisoblanadi agar identifikator aniqlanmagan bo'lsa. Dasturga ulash muljallangan fayllarning har biriga bitta fayl ulanish mo'ljallangan bo'lsa, bu fayl bir necha marta dasturga ulanib koladi. Bu qayta ulanishni oldini olish uchun standart fayllar yuqorida ko'rilgan direktivalar yordamida himoya qilingandir. Bu himoya usuli qo'yidagicha bo'lishi mumkin.

```
/* filename Nomli fayl */
/* FILENAME aniklanganligini tekshirish */
# indef FILE_NAME
... /* Ulanaetgan fayl teksti
    /* Ta'rif
#define FILE_NAME
#endif
```

Tarmoqlanuvchi shartli direktivalar yaratish uchun qo'yidagi direktiva kiritilgan:

#elif butun_sonli_ifoda Bu direktiva ishlatilgan tekst strukturasi:

```
#if shart
tekst
#elif 1_ifoda
1_tekst
#elif 2_ifoda
2_tekst
...
#else
tekst
#endif
```

Preprotsesssor avval #if direktivasidagi shartni tekshiradi. Agar shart 0 ga teng bulsa 1_ifoda hisoblanadi agar u ham 0 bulsa 2_ifodani hisoblaydi va hokazo. Agar hamma ifodalar 0 bulsa else uchun kursatilgan tekst ulanadi. Agar biror ifoda 0 dan katta bulsa shu direktivada kursatilgan tekst ulanadi.

48 - DARS.DEFINED OPERATSIYASI.

Tekst shartli qayta ishlanganda unar preprotsessor amali Defined operand amalidan foydalanish mumkin. If defined ifodasi #ifdef operand ifodasiga ekvivalentdir. Bu ko'rinishda defined avfzalligi bilinmaydi. Misol uchun biror tekst kompilyatorga Y identifikatori aniqlangan, N bo'lsa aniqlanmagan holda uzatish lozim bo'lsin. U holda preprotsessor direktivasi qo'yidagicha yoziladi:

```
#if defined Y&&!defined N
tekst
#endif
```

Bu direktivani qo'yidagicha ham yozish mumkin.

```
#ifdef Y
#ifndef N
tekst
#endif
#endif
```

Yordamchi direktivalar.

Satrlarni nomerlash uchun quyidagi direktivadan foydalanish mumkin:

```
#line konstanta
```

Direktiva fakat satr nomeri emas, fayl nomini ham uzgartirishi mumkin:

```
 #line konstanta "fayl nomi"
```

Odatda bu direktiva kam ishlatiladi.

Quyidagi direktiva leksemalar ketma ketligi orqali kursatilgan shaklda diagnostik ma'lumotlar berilishiga olib keladi.

```
# error leksemalar ketma ketligi.
```

Misol uchun NAME preprotsessor o'zgaruvchisi aniqlangan bo'lsin:

```
#define NAME 5
```

Dasturda bu o'zgaruvchi qiymatini teshirib, 5 ga teng bo'lmagan holda ma'lumot berish uchun qo'yidagi direktivadan foydalaniladi:

```
 #if (NAME!=5)
#error NAME 5 ga teng bo'lishi kerak
```

Hech qanday hizmat bajarmaydigan direktiva: #

49 - DARS. MAKROSLAR.

Makros ta'rifiga ko'ra bir simvollar ketma – ketligi bilan almashtirishdir. Eng sodda makro ta'rif

define identifikator almashtiruvchi satr.

Bu direktiva yordamida foydalanuvchi asosiy tiplar uchun yangi nomlar kiritishi mumkin.
Masalan: # define Real Long double
Dastur matnida Long double tipidagi o'zgaruvchilarni Real sifatida ta'riflash mumkin.
Masalan: # define Range((int _ Max)-(int _ Min)+1)
Parametrli makrota'riflardan foydalanish yanada kengroq imkoniyatlar yaratadi:
define nom (parametrlar ruyhati) almashtiriluvchi_qator
Bu erda nom – makros nomi.
Parametrlar ruyhati – vergul bilan ajratilgan identifikatorlar ruyhati.
Makrota'rifning klassik misoli :

```
# define max (a,b)  (a<b ? b:a)
```

Bu makrosdan foydalanganda kompilyator max (a,b) ifodani (x<a ? y:x) ifoda bilan almashtiradi.
Yana bir klassik misol:

```
# define ABS(x) (x<0 ? –(x):x)
```

Misol uchun dasturdagi ABS(E-Z) ifoda (E-Z<0 ? (E-Z):E-Z) ifoda bilan almashtiriladi.

Makroslar ko'p o'lchovli massivlar bilan ishlashda yordam beradi. Matritsalar bilan ishlaganda quyidagi chegaralar mavjud. Jadvallarning birinchi indeksidan boshqa hamma indekslari elementlari soni ko'rsatilishi shart. Massivlar elementlari nomerlari 1 dan emas 0 dan boshlanadi.

Birinchi cheklanishdan kutulish yuli matritsa o'rniga bir o'lchovli massiv kiritish , lekin bu massiv bilan matritsa shaklida amallar bajarish. Bu vazifani makros bajarish imkoniyatini beradi. Makros ikkinchi chegarani engish imkonini ham beradi.

Misol:

```
# define N4
# define M5
# define A(I,j)  x [M(I-1)+(j-1)]
# include <iostream.h>
void main ()
{ double x[NM];
int I, j, k;
for (k=0 ; k<NM ; k++)
x[k]=k;
for (I=1; I<=N; I++)
{ Cout<< (“\n stroka%d: “,i);
for (j=1; j<=M; j++)
Cout<< (“% 6.1f “, A(I,j));
}
}
```

Dasturda Cout<< (“% 6.1f “, A(I,j)); ifoda makros joylashdan so'ng quyidagi ko'rinishga keladi.

```
Cout<< (“% 6.1f “, x[5(I-1)+(y-1)]);
```

Almashtiruvchi qatorda preprotsessor amallari.

Almashtiruvchi qatorni tashkil qiluvchi leksemalar ketma ketligida ‘#’ va ‘##’ amallarini qo’llash mumkin. Birinchi amal parametr oldiga qo’yilib, shu parametrni almashtiruvchi qator qavslarga olinishi kerakligini ko’rsatadi. Misol uchun:

#define print(A) print(#A”=%f”,A)

makro ta'rif berilgan bo’lsin. U holda makrosga print(a+b) murojaat quyidagi makro kengaytmani hosil kiladi: print(“a+b””%f”,a+b).

Ikkinchi ‘##’ amal leksemalar orasida qullanilib, leksemalarni ulashga imkon beradi. Quyidagi makrota'rif berilgan bulsin:

#define zero(a,b,c) (bac)
#define one(a,b,c) (b a c)
#define two(a,b,c) (b##a##c)

Makrochakirik:	Makrojoylash natijasi:
Zero(+,x,y)	(bac)
One(+, x, y)	(x + y)
Two(+,x,y)	(x+y)

Birinchi holda bas yagona identifikator deb qaralib makro almashtirish amalga oshirilmaydi. Ikkinchi holda makros argumentlari bo’shlik belgilari Bilan ajratilgan bo’lib bu belgilar natijada ham saqlanib koladi. Uchinchi holda makros uchun ‘##’ amali qo’llanilgani uchun natijada bo’shlik belgilarsiz parametrlar ulanadi.

Makroslarning funktsiyadan farki.

Funktsiya dasturda bitta nushada bo’lsa, makros hosil qiluvchi matnlar makros har gal chaqirilganda dasturga joylashtiriladi. Funktsiya parametrlar spetsifikatsiyasida ko’rsatilgan tiplar uchun ishlatiladi va konkret tipdagi qiymat qaytaradi. Makros har qanday tipdagi parametrlar bilan ishlaydi. Hosil qilinayotgan qiymat tipi faqat parametrlar tiplari va ifodalarga boglik. Makrojoylashlardan tug’ri foydalanish uchun almashtiriluvchi satrni qavsga olish foydalidir. Funktsiyaning haqiqiy parametrlari bu ifodalardir, makros argumentlari bo’lsa vergul bilan ajratilgan leksemalardir. Argumentlarga makro-kengaytirishlar qo’llanmaydi.

Oldindan kiritilgan makronomlar.

..LINE.. – o’nlik konstanta o’qilayotgan satr nomeri . Birinchi satr nomeri 1 ga teng.
..FINE.. – fayl nomi . simvollar qatori. Preprotsessor har gal boshqa fayl nomi ko’rsatilgan # include direktivasini uchratganda nom o’zgaradi. # include direktivasi bajarilib bo’lgandan so’ng nom qayta tiklanadi.
..DATE.. – “Oy , sana, yil” formatidagi simvollar satri. Fayl bilan ishlash boshlangan sanani ko’rsatadi.
..TIME.. – “Soatlar : minutlar : sekundlar ” formatidagi simvollar satri. Preprotsessor tomonidan faylni o’qish boshlangan vaqtni ko’rsatada.
..STDC.. – Agar kompilyator ANSI – standart bo’yicha ishlayotgan bo’lsa qiymati 1 ga teng konstanta. Aks holda konstanta qiymati aniqlanmagan.
Borland C++ kompilyatori preprotsessorida qo’shimcha konstantalar kiritilgan:
..BCOPT.. – agar kompilyatsiyada optimizatsiya ishlatilsa qiymati 1 ga teng konstanta.
..BCPLUSPLUS.. – kompilyator versiyasiga mos keluvchi son qiymati.
..CPECL.. – funktsiyalarga parametrlar uzatish tartibini belgilaydi. C ++ da qabul qilingan tartib 1 raqamga tug’ri keladi.
.. CONSOLE.. – 32 razryadli kompilyator uchun aniqlangan va konsal programmalar uchun 1 ga teng.

..DLL.. – WINDOWS DLL rejimida ishlashga mos keladi.
..MSDOS.. -- Borland C++ 16 razryadli kompilyatorlari uchun 1 ga teng va 32 razryadli kompilyator uchun 0 ga teng.
..MT.. – Makros faqat 32 razryadli kompilyatorlar uchun mo'ljallangan.
..OVERLAY.. – overleyi rejimida 1 ga teng.
..PASCAL.. – CDECL ga qarama-qarshi.
..TCPLUSPLUS.. – kompilyator versiyasiga mos keluvchi son qiymati. C++ kompilyatorlari uchun mo'ljallangan.
..TLS.. -- 32 razryadli kompilyatorlar uchun rost deb belgilangan.
..TURBOC.. -- Borland C++ 4.0 kompilyatori uchun 0X0 400 kiymatga teng(Borland C++ 4.5 uchun 0X0 460 va hakazo).
.. WINDOWS.. – WINDOWS uchun kod generatsiyasi .
.. WIN32.. – 32 razryadli kompilyator uchun aniqlangan va konsol dasturlar uchun 1 ga teng.

50 - DARS. MA'LUMOTLARNI KIRITISH VA CHIQARISH.

Qo'yidagi funktsiyalar dasturda simvollarni kiritish va chiqarish uchun ishlatiladi.

getch(arg) – bitta simvol kiritilishini kutish. Kiritilayotgan simvol monitorda aks etmaydi. Bu funktsiyani programma ohirida argumentsiz ishlatilsa, monitorda ma'lumotlarni to klavisha bosilguncha o'qish mumkin bo'ladi.

putch(arg)- bitta simvolni standart okimga chikarish uchun ishlatiladi.Simvol monitorda aks etmaydi.

getchar(arg) – bitta simvol kiritilishini kutish. Kiritilayotgan simvol monitorda aks etadi. Bu funktsiyani programma ohirida argumentsiz ishlatilsa, monitorda ma'lumotlarni to klavisha bosilguncha o'qish mumkin bo'ladi.

putchar(arg) - bitta simvolni standart oqimga chiqarish uchun ishlatiladi.Simvol monitorda aks etadi. Bu funktsiyalar iostream.h modulida joylashgandir.

Misol:

```
Include <iostream.h>
Void main()
{ int c;
c=getchar();
putchar(c);
c=getch();
putchar();
getch();
}
```

51 - DARS. FORMATLI CHIQARISH – PRINTF.

Printf funktsiyasi ko'rsatilgan parametrlarni standart oqimga chiqarish uchun ishlatiladi. Standart oqim tushunchasi keyingi boblarda yoritiladi. Hozircha standart oqim sifatida monitor tushunilishi etarlidir. Funktsiya iostream.h modulida joylashgan bo'lib, umumiy ko'rinishi qo'yidagichadir:

Printf(control,arg1,arg2,...)

Control boshqaruvchi qator deb atalib ikki turdagi simvollardan iborat bo'ladi: oddiy chiqariluvchi simvollar va navbatdagi parametrni o'zgartirib chiqaruvchi spetsifikatsiyalar. Har bir spetsifikatsiya % simvolidan boshlanib o'zgartirish turini ko'rsatuvchi simvol bilan tugaydi. % belgisi va o'zgartirish simvoli orasiga qo'yidagi simvollarni qo'yish mumkin. Chiqarilayotgan argument chapga tekislash lozimligini ko'rsatuvchi minus belgisi. Maydon minimal uzunligini ko'rsatuvchi raqamlar qatori. Maydon uzunligini keyingi raqamlar qatoridan ajratuvchi nuqta.

Biror qatordan qancha simvol ajratib olish lozimligini hamda float yoki double tipidagi sonlarda nuqtadan keyin qancha kasr raqamlari bosib chiqarilishini ko'rsatuvchi raqamlar ketma-ketligi.

Chiqarilaetgan son long tipiga tegishli ekanligini ko'rsatuvchi uzunlik markeri l.

O'zgartirish simvollari qo'yidagilardan iborat.

d– parametr unli butun songa aylantiriladi.

o – parametr ishorasiz va birinchi rakami 0 bo'lmagan sakkizlik songa aylantiriladi.

x – parametr ishorasiz va 0x belgisiz un oltilik songa aylantiriladi.

h - parametr ishorasiz o'nlik songa aylantiriladi.

c – parametr bitta simvol deb qaraladi.

s – parametr satr simvollar nulinchi simvol uchramaguncha yoki ko'rsatilgan sondagi simvollar bosiladi

e – parametr float eki double tipidagi son deb qaraladi va ishorali m.nnnnnnE+-xx ko'rinishidagi o'nlik songa keltiriladi.

f - parametr float yoki double tipidagi son deb qaraladi va ishorali m.nnnnnn ko'rinishidagi o'nlik songa keltiriladi.

g - %e eki %f sifatida ishlatiladi.

% dan keyingi simvol o'zgartirish simvoli bo'lmasa u bosmaga chiqariladi.

% simvolini o'zini bosmaga chiqarish uchun %% belgisini berish lozim.

Quyidagi jadval har hil spetsifikatsiyalarni "HELLO, WORLD" (12 simvolov) so'zini bosishga ta'sirini ko'rsatadi. Bu erda har bir maydon uzunligini ko'rsatish uchun maydon ohiriga ikki nuqta qo'yilgan.

```
:%10S:        :HELLO, WORLD:
:%10-S:       :HELLO, WORLD:
:%20S:        :    HELLO, WORLD:
:%-20S:       :HELLO, WORLD      :
:%20.10S:     :      HELLO, WOR:
:%-20.10S:    :HELLO, WOR       :
:%.10S:       :HELLO, WOR:
```

52 - DARS. FORMATLI KIRITISH – SCANF.

Scanf funktsiyasi iostream.h modulida joylashgan bo'lib, umumiy ko'rinishi qo'yidagichadir:

Scanf(control, arg1, arg2,...)

Funktsiya standart okimdan simvollarni o'qib boshqaruvchi qator asosida formatlab mos parametrlarga yozib qo'yadi. Parametr ko'rsatkich bo'lishi lozim. Boshqaruvchi qator qo'yidagi o'zgartirish spetsifikatsiyalaridan iborat Bushlik, tabulyatsiya, keyingi qatorga o'tish simvollari;

Oddiy simvollar (% dan tashkari) kiritish oqimidagi navbatdagi simvollar bilan mos kelishi lozim;

% simvolidan boshlanuvchi spetsifikatsiya simvollari;

% simvolidan boshlanuvchi qiymat berishni ta'qiqlovchi * simvoli;

% simvolidan boshlanuvchi maydon maksimal uzunligini ko'rsatuvchi son;

Qo'yidagi spetsifikatsiya simvollarini ishlatish mumkin:

d – unli butun son ko'tilmoqda.

o – 0 bilan boshlangan yoki boshlanmagan sakkizlik son kutilmoqda.

x – 0x belgili eki belgisiz o'n oltilik son kutilmoqda.

h - o'nlik son kutilmoqda.

c – bitta simvol kutilmoqda.

s – satr kutilmoqda.

f - float tipidagi son kutilmoqda. Kiritilayotgan sonning butun raqamlari va nuqtadan so'ng kasr raqamlari soni va E eki e belgisidan so'ng mantissa raqamlari soni ko'rsatilishi mumkin.

53 - DARS. FAYLLAR. OQIMLI KIRITISH VA CHIQARISH.

C ++ tilining asosiy hususiyatlaridan biri oldindan rejalashtirilgan fayllar strukturasi yukligidir. Hamma fayllar, baytlar ketma-ketligi deb ko'riladi. U N I X operatsion sistemasida har bir qurilmaga «Mahsus fayl» mos keladi, shuning uchun C ++ bibliotekasidagi funktsiyalar fayllar bilan ham, qurilmalar bilan ham ma'lumot almashinishi uchun foydalaniladi. C ++ tili bibliotekasida kiritish – chiqarish, quyi darajadagi kiritish, chiqarish va portlar uchun kiritish – chiqarish, oqimli daraja tizim hususiyatlariga bog'lik bulishi uchun bu erda qaralmaydi.
Oqimli chiqarish va kiritishda ma'lumotlar bilan almashish baytma-bat amalga oshiriladi. Lekin tashki hotira qurilmalari bilan almashish oldidan belgilangan ma'lumotlar bloki orqali amalga oshiriladi odatda u blokning minimal hajmi 512 yoki 1024 baytga teng bo'ladi. Diskga o'qilishda ma'lumotlar operatsion qatordagi buferi yoziladi so'ngra baytma bayt buferga yig'iladi, so'ngra diskka har bir murojaat qilinganda yagona blok sifatida uzatiladi. Shuning uchun ma'lumot almashishi diskka to'g'ridan to'g'ri murojaat qilishiga ko'ra tezroq amalga oshadi. Shunday qilib oqim bu bu buferlash vositalari va fayldir.
Oqim bilan ishlashda qo'yidagi vazifalarni bajarish mumkin.
- Oqimlarni ochish va yopish.
- Simvol, qator satr ,formatlangan ma'lumot ihtiyoriy uzunlikdagi ma'lumotlarni kiritish yoki chiqarish va fayl ohiriga etganlik shartini tahlil qilish;
- Buferlash va bufer hajmini boshqarish;
- Ko'rsatkich oqimdagi o'rnini aniqlash yoki yangi o'ringa ko'chirish.

Bu vazifalarni boshqaruvchi funktsiyalar teng foydalanish dasturiga Stdio.h – faylini ulash lozim.
Dastur bajarilishi boshlanganda avtomatik ravishda 5 ta oqim ochilib, bulardan:
- Standart kiritish oqimi stdin;
- Standart chiqarish oqimi stdout;
- Hatolar haqida malumotlar standart oqimi stderr;

Oqimlarni ochish va yopish

Oqim ochilishi uchun, oldindan kiritilgan FILE tipidagi struktura bilan boglash lozimdir. FILE strukturasi ta'rifi iostream.h bibleotekasida joylashgan. Bu strukturada buferga ko'rsatkich, o'qilayotgan pozitsiyaga ko'rsatkich va boshqa ma'lumotlar saqlanadi. Oqim ochilganda dasturda oqimga ko'rsatkich ya'ni FILE strukturali tipdagi ob'ektga ko'rsatkich qaytariladi. Bu ko'rsatkich qo'yidagicha e'lon qilinishi lozim.
FILE * <kursatkich nomi>
Misol uchun FILE * fp
Oqim ochish funktsiyasi quyidagi ko'rinishga ega;
<oqimga ko'rsatkich nomi>=foren(<fayl-nomi>,<ochish rejimi>)
Misol uchun:fp=fopen("t.tnt", "r")
Oqim bilan bog'lik faylni qo'yidagi rejimlarda ochish mumkin:
" w"- Yangi fayl o'qish uchun ochiladi. Agar fayl mavjud bo'lmasa yangidan yaratiladi.
"r" - Mavjud fayl faqat o'qish uchun ochiladi.
"a" - Fayl da'vom ettirish uchun ochiladi.
"wt" - Fayl yozish va keyingi tahrirlash uchun ochiladi. Fayl ihtiyoriy joyidan o'qish yoki yozish mumkin.

“rt”- fayl ihtiyoriy joyidan o’qish yoki yozish mumkin, lekin fayl ohiriga qo’shish mumkin emas.
“at” - Fayl ihtiyoriy joyidan o’qish va yozish uchun ochiladi “wt” rejmdan farqli fayl ohiriga ma'lumot qo’shish mumkin.
Oqimdan o’qilgan qo’yidagi simvollar -------
CR(13)-naryat nomi qaytarish
RF(10)-“yangi qator” boshiga o’tish bitta simvolga “\n” (10) fqkfynbhbkflb/
Agar fayl------- emas ihtiyoriy bulsa, binar rejimda ochiladi. Buning uchun rejimlar---------- harfi qo’shiladi ------- “wb” yoki “rtb”. Ba'zi ----- matnli rejim t harifi yordamida ko’rsatiladi masalan “yoki”rt”.
Oqim ochilganda quyidagi hatolar kelib chiqishi mumkin:ko’rsatilgan fayl mavjud emas(o’kish rejimida); disk to’la yoki yozishdan himoyalangan va hokazo. Yana shuni aytish kerakki fopen() funktsiyasi bajarilganda dinamik hotira ishlatiladi. Agar hotirada joy qolmagan bo’lsa “not enough ” - hatosi kelib chiqadi.
Ko’rsatilgan hollarda ko’rsatkich ~ NULL qiymatga ega bo’ladi.
Bu hatolar haqidagi ma'lumotlarni ekranga chiqarish uchun perror () funktsiyasi ishlatiladi. Bu funktsiya iostream.h bibliotekasida saqlanuvchi prototipi qo’yidagi ko’rinishga ega.:
Void perror(court char * s);
Diskda ochilgan fayllarni berkitish uchun qo’yidagi funktsiyadan foydalaniladi.
Int fellove(<oqimga-kursatkich nomi>).
Fayllar bilan ishlashning bitli rejimi.
Fayl bilan bitli almashish rejimi getc( ) va putc( ) funktsiyalari yordamida tashkil etiladi. Bu funktsiyalarga qo’yidagi shaklda murojat etiladi:

```
C=getc(fp);
Putc(c,fp);
```

Bu erda fp-ko’rsatkich
S-int tipidagi o’zgaruvchi
Misol tariqasida klaviaturadan simvol kiritib faylga yozishni ko’ramiz. Matn ohirini ‘#’ belgisi ko’rsatadi. Fayl nomi foydalanuvchidan so’raladi. Agar <enter> klavishasi bosilsa faylga CR va LF (qiymatlari 13 va 10) konstantalar yoziladi. Keyinchalik fayldan simvollarni uqishda bu konstantalar satrlarni ajratishga imkon beradi.

```
#include <iostream.h>
int main()
{ file *fp;
char c;
const char CR=’\015’;
const char LF=’\012’;
char f name [20];
puts(“fayl nomini kiriting:\n”);
gets(f name);
if((fp=f open(f name, “w”)) ==null)
{ perror(f name);
return 1;
}
while ((c=getchar())!=’#’)
}
if (c==’\n’)
{ putc(CR,fp);
putc(LF,fp);
```

```
}
else putc (c,fp);
}
Fclose (fp);
Return 0;
```
Keyingi programma fayldan simvollarni o’qib ekranga chiqaradi.
```
#include <iostream.h>
int main()
{ file *fp;
char c;
char f name [20];
puts(“fayl nomini kiriting:\n”);
if((fp=f open (f name, “r”)) ==null)
{ perror(f name);
return 1;
}
while ((c=getc(fp))!=eof)
putchar(c);
f close (fp);
return 0;
}
```
Satrlar yordamida fayllar bilan bog’lanish.
Matnli fayllar bilan ishlash uchun fget va fputs funktsiyalaridan foydalaniladi. Bu funktsiyalari prototiplari iostream.h faylida qo’yidagi ko’rinishga ega:
```
Int fputs (const char *s, FILE *stream);
Char *fgets (char * s, int n, FILE * stream);
```
Fputs funktsiyasi ‘\0’ simvoli bilan chegaralangan satrni stream ko’rsatkichi orqali aniqlangan faylga yozadi. ‘\0’ simvoli faylga yozilmaydi.
Fgets() funktsiyasi stream ko’rsatkichi orqali aniqlangan fayldan (n-1) simvolni o’qiydi va S ko’rsatgan satrga yozib qo’yadi. Funktsiya n-1 simvolni o’qib bo’lsa eki 1-chi qator simvoli ‘\n’ni uchratsa ishini tuhtatadi. Har bir satr ohiriga qo’shimcha \0 belgisi qo’shiladi. Hato bo’lganda yoki fayl ohiriga etganda agar fayldan birorta simvol o’qilmagan bo’lsa NULL qiymat qaytariladi. Qo’yidagi dasturda bir fayldagi matnni ikkinchi faylga yozishni ko’rib chiqamiz. Bu misolda yana bir imkoniyat komanda qatoridan programmaga ma'lumot uzatish imkoniyati ko’rib chiqilgan. Har qanday dastur operatsion sistemada ma'lumotni argc va argv parametrlar qiymati sifatida oladi. Birinchi programmaga uzatilayotgan satrlar sonini ko’rsatadi. Argv[0] bu faylning nomini saklovchi satrga ko’rsatkich massivining qolgan elementlari argv[10]...argv[argc-1] komanda qatorida fayl nomidan so’ng bo’shlik tashlab yozilgan parametrlarga ko’rsatkichlar.
Programmamiz nomi coryfile.exe bo’lsin va bu programma yordamida f1.dat Faylni f2.dat faylga yozmoqchimiz. Komanda qatori qo’yidagi ko’rinishga ega:
<copyfile.exe f1.dat f2.txt
Programma matni:
```
#include <iostream.h>
main (int argc, char*argv[])
{ char cc[256];
FILE *f1, *f2;
If (argc!=3)
{ print (“\n Format bazovih programma:”);
```

```
print f (“\n copyfile.exe”)
Cout<< (“\n Fayl netosnihh Fayl priemnik”);
return 1;
}
if ((f1=fopen(argv[1],”r”))==NULL)
{perror(argv[1]);
return 1;
}
if ((f2=fopen(argv[2], “w”))==NULL)
{perror(arg[2]);
return 1;
}
while (fgets(cc,256,f1)!=NULL)
fputs(CC,f2);
fclose(f1);
fclose(f2);
return 0;
}
```

Bu dastur bajarilishi natijasida int.dat fayliga Cout<< funktsiyasi yordamida monitorga qanday chiqsa shunday ko’rinishda ma'lumotlar yozadi. Keyingi misolda fayldan monitorga o’qishni kuramiz:

```
#include <iostream.h>
int main()
{
FILE *fp;
Intn,nn,I;
If((fp=fopen(“int.dat”,”r”))==NULL)
{perror (“int.dat”);
return 1;
}
for(i=1; i<11;i++)
{fCin>>(fp,”%d”,&n)        ;
Cout<<(“%d \n”,n);
}
fclose(fp);
return 0;
}
```

54 - DARS. FAYLLAR BILAN FORMATLI ALMASHINUV.

Ko’p hollarda ma'lumotni tug’ridan-tug’ri monitorga chiqarishga qo’lay shaklda faylda saqlash zarur bo’ladi. Bu holda faylga formatli kiritish va chiqarish funktsiyalaridan foydalanish mumkin. Bu funktsiyalar qo’yidagi prototiplarga ega: Int fprint(oqimga ko’rsatkich, formatlash-qatori, o’zgaruvchilar ro’yhati); Int fCin>>(oqimga ko’rsatkich, formatlash-qatori, o’zgaruvchilar ro’yhati);
Misol tariqasida int .dat faylini yaratuvchi va bu faylga 1 dan 100 gacha bo’lgan sonlarning simvolli tasvirini yozib qo’yuvchi programmani ko’rib chiqamiz:

```
#include <iostream.h>
int main()
{
FILE *fp;
Int n;
If((fp=fopen(“int.dat”,”ts”))==NULL)
{perror (“int.dat”);
return 1;
}
for(n=1; n<11;n++)
fCout<<(fp,“%d ”;n);
}
fclose(fp);
return 0;
}
```

55 - DARS. FAYLGA IHTIYORIY MUROJAT QILISH

Hozirgi ko’rib chiqilgan funktsiyalar faylga ketma-ket yozish yoki ketma-ket uqishga imkon beradi holos. Fayldan uqib faylga yozishlar doim joriy pozitsiyasida bo’ladi. Boshlang’ich pozitsiya fayl ochilganda aniqlanadi. Faylni "r" va"w" rejimida ochilganda joriy pozitsiya ko’rsatgichi faylning birligi baytini ko’rsatadi, "a" rejimida ochilganda, oshish baytini ko’rsatadi. Har bir kiritish-chiqarish amali bajarilganda, ko’rsatgich o’qilgan baytlar soniga qarab yangi pozitsiyaga ko’chadi. Faylning ihtiyoriy baytiga murojat qilish uchun fseek() funktsiyasidan foydalanish lozimdir.Bu funktsiya qo’yidagi prototipga ega.

Int fseek (faylga ko’rsatgich, oraliq,hisobat boshi) farq log tipidagi o’zgaruvchi yoki ifoda bilan beriladi. Hisobot boshi oldin qo’yidagi konstantalardan biri bilan aniqlanadi.

Seek_ Set (qiymati 0)-fayl boshi;

Seek_cur (qiymati 1)-uqilayotgan pazitsiya;

Seek_ end (qiymati 2)-fayl ochish;

Fseek () funktsiyasi 0 qaytaradi agar faylda ko’chish bajarilgan bo’lsa, aksincha noldan farqli songa teng bo’ladi.

Ihtiyoriy pozitsiyadan fayl boshiga o’tish:

Fseek (fp,ol,seek-set)

Ihtiyoriy pozitsiyadan fayl boshiga o’tish:

Fseek (fp,ol,seek-end)

Joriy pozitsiyadan bir bayt oldinga yoki orqaga ko’chish uchun fseek (jp,-1L,seek-cur).

Fseek funksiyasidan tashqari C ++ tili bibliotekasida pozitsiyaga ko’rsatkichlar bilan bog’lik qo’yidagi funktsiyalar mavjud.

Long ftell (FIHE*)-faylda ko’rsatkichning joriy pozitsiyasini aniqlash.

Void rewind (FILE*)-joriy pozitsiya ko’rsatkichini fayl boshiga keltirish.

Qo’yi darajadagi kiritish va chiqarish.

Qo’yi darajadagi kiritish va chiqarish funtsiyalari operatsion tizim imkoniyatlaridan to’g’ridan to’g’ri foydalanishga imkon beradi. Bu holda buferlash va formatlash bajarilmaydi. Faylni qo’yi darajadagi ochishda fayl bilan fayl (oqim) ko’rsatkichi emas, deskriptor bog’lanadi. Fayl deskriptori fayl ochilganligi to’grisidagi ma'lumotni operatsion tizim ichki jadvallariga joylashtiruvini belgilovchi butun sondir. Qo’yi darajadagi funktsiyalar dasturga iostream.h bibliotekasini qo’shishni talab qilmaydi. Lekin bu biblioteka fayllar bilan ishlashda foydali bo’lgan ba'zi konstantalar (misol uchun fayl yakuni belgisi EOF) tarifini uz ichiga oladi. Bu konstatalarda foydalanganda iostream.h dasturga qo’shilishi zarurdir.

56 - DARS. FAYLLARNI OCHISH VA YOPISH.

Fayllarni qo'yi darajadada ochish uchun open () funktsiyasidan foydalaniladi:
int fd= open (fayl nomi, bayroqlar, murojat.)
fd – fayl deskriptori,
fayl nomi – simvollar massiviga ko'rsatkichdir.
2- parametr bayroqlar fayl ochish rejimini belgilovchi ifodadir. Bu ifoda fcntl.h sarlavhali faylda saqlanuvchi konstantalardan biri yoki shu kostantalardan razryadli '|' amali yordamida hosil qilingan bo'lishi mumkin.
Kostantalar ro'yhati:

O_APPEND Faylni ohiriga yozuv qo'shish uchun ochish;

O_BINARY Faylni bitli (ikkili)binar rejimda ochish

O_CREAT Yangi fayl yaratish va ochish

O_EXCL Agar O_CREAT bilan birga ko'rsatilgan bo'lsa va yaratilmoqchi bo'lgan fayl mavjud bo'lsa faylni ochish funktsiyasi hatolik bilan tugaydi. Mavjud faylni o'chib ketmaslikdan saqlaydi.

O_RDONLY Faylni faqat o'qish uchun ochish

O_RDWR Faylni o'qish va yozish uchun ochish

O_TEXT Faylni tekstli rejimda ochish

O_TRUNK Mavjud faylni ochish va bor ma'lumotni o'chirish

Fayl ochilish rejimi albatta ko'rsatilgan bo'lishi shart. 3- parametr murojat huquqlari faqat faylni O_CREAT ochish rejimida ya'ni yangi fayl yaratishda foydalaniladi. MS DOS va MS WINDOWS operatsion tizimlarida murojat huquqlari parametrlarini berish uchun quyidagi konstantalardan foydalaniladi.

S_IWRITE Faylga yozishga ruhsat berish

S_IREAD Fayldan uqishga ruhsat berish

S_ IREAD\ S_WRITE Uqish va yozishga ruhsat berish

Ko'rsatilgan konstantalar sys katalogida joylashgan stat.h sarvlahali faylda saqlanadi. Bu faylni qo'shish # include <sys\stade.h> direktivasi orqali amalga oshiriladi. Agar murojaat huquqi parametri ko'rsatilmagan bo'lsa faqat fayldan o'qishga ruhsat beriladi.
UNIX operatsion tizimida murojaat huquqlari 3 hil foydalanuvchilar uchun ko'rsatiladi:
Fayl egasi;
Foydalanuvchilar guruhi a'zosi.
Boshqa foydalanuvchilar
Foydalanuvchilar huquqlari quyidagi simvollar orqali ko'rsatiladi:
R- fayldan uqish ruhsat berilgan.
W- faylga yozish ruhsat berilgan.
X- fayllarni bajarish ruhsat berilgan.

Agar biror murojaat huquqi berilmagan bo'lsa urniga `_` belgisi qo'yiladi. Agar fayl egasiga hamma huquqlar, foydalanuvchi guruhi a'zolariga o'qish va bajarish, boshqa foydalanuvchilarga faqat bajarish huquqi berilgan bo'lsa, murojaat qatorini quyidagicha yozish mumkin rwxr-x—x. Har bir `_` simvol urniga 0 rakami, aks holda 1 raqami qo'yilib hosil bo'lgan sondagi o'ng tomondan boshlab har bir uch raqamini sakkizlik son sifatida yozilsa, murojaat huquqini belgilovchi sakkizlik butun son hosil bo'ladi. Yuqorida hosil qilingan rwxr-x—x qatori ikkilik 111101001 nihoyat sakkizlik 0751 son shaklida yozilib open ( ) funktsiyasida murojaat huquqi parametri sifatida ko'rsatiladi. Faylni ochishga misollar:
1. faylni o'qish uchun ochish:
fd=open (" t.txt " , O_RONLY)

2. faylni o'qish va yozish uchun ochish:
fd = open(" t.txt " , O_RDWR)

3. faylni yangi ma'lumotlar yozish uchun ochish:
fd = open(" new.txt " ,O_WRONLY_ |O-Creat| O_TRUNK, 0600)

Sakkizlik konstanta 0600 shaklida berilgan murojaat huquqi parametrining simvolli ko'nishi rw -------- bulib, fayl egasiga o'qish va yozish huquqi , qolgan foydalanuvchilarga hech qanday huquq berilmaganligini bildiradi . Faylni ochishda kelib chiqadigan hato turini aniqlash uchun errno.h sarlavhali faylda saqlanuvchi errno o'zgaruvchisi hizmat qiladi. Agar bu o'zgaruvchi qiymati shu sarlavhali faylda saqlanuvchi EEXIST konstantasiga teng bo'lsa ochilayotgan fayl mavjudligini bildiradi. Sopen ( ) funktsiyasi bitta faylga bir necha dasturlardan murojaat qilish imkonini beradi. Albatta dasturlar faylga faqat o'qish rejimida murojaat qilishi mumkin. Faylni ochish uchun yana Creat ( ) funktsiyasi mavjud bulib qo'yidagi Open ( ) funktsiyasini chaqirishga mos keladi.
Open (fayl nomi, O_creat |O_TRUNK| O_WRONLY); bu funktsiya yangi fayl yaratadi va yozish uchun ochadi. Qo'yi darajada fayllarni yopish uchun close ( ) funktsiyasidan foydalanish lozim. Bu funktsiya ko'rinishi qo'yidagichadir:
Int close (fayl deskriptori). Funktsiya muvofaqiyatli bajarilganda 0 qaytaradi. Hato bo'lganda – 1.
Ma'lumotlarni o'qish va yozish
Qo'yi darajada ma'lumotlarni kiritish va chiqarish read ( ) va write ( ) funktsiyalari orqali amalga oshiriladi. Bu funktsiyalar prototiplari qo'yidagi ko'rinishga ega:
int read (int fd, char * buffer; unrigned int count)
int write (int fd, char * buffer; unsiqned int count)
Ikkala funktsiya butun o'qilgan yoki yozilgan baytlar sonini qaytaradi. Read funktsiyasi fd deskriptori bilan ochilgan fayldan count parametrida ko'rsatilgan miqdordagi baytlarni o'qib, buffer ko'rsatkichi orqali ko'rsatilgan bufferga yozadi. Fayl ohiriga etganda read () funktsyasi 0 qiymat qaytaradi. Fayldan o'qishda hatolik kelib chiqsa -1 qiymat qaytaradi. O'qish fayldagi joriy pozitsiyadan boshlanadi. Agar fayl matnli rejimda ochilsa CR va LF simvollari `\n` simvoliga o'zgartiriladi.
Write ( ) funktsiyasi fd deskriptori bilan ochilgan faylga buffer ko'rsatkichi orqali ko'rsatilgan bufferdan count parametri orqali ko'rsatilgan miqdordagi baytlarni yozib qo'yadi. Yozuv joriy pozitsiyadan boshlanadi. Agar fayl matnli rejimda ochilgan bo'lsa`\n` simvolini CR va LF simvollar sifatida yoziladi. Agar yozishda hatolik kelib chiqsa, write ( ) funktsiyasi -1 qymat qaytaradi. Errno global o'zgaruvchisi bo'lsa Errno.h sarlavhali faylda ko'rsatilgan qo'yidagi konstantalar biriga teng buladi.

EACCES – fayl yozuvdan himoyalangan

ENOSPC – tashki kurilmada bush joy kolmagan

EBADF – notugri fayl deskriptori

Bu funktsiyalar io.h sarlavhali faylda joylashgandir. Qo'yida bir fayldan ikkinchisiga nusha olish dasturini ko'rib chiqamiz:

programma

```
#include <iostream.h>
#include <fcnfl.h>
#include <io.h>
int main(int argc,  char *argv[ ] )
{
 int  fdin , fdout;  /*Deskriptorih  faylov*/
 int n; /* Kolichestvo prochitannihh  baytov*/
 char buff[BUFSIZ];
 if  (argc !=3)
  {
  Cout<< ("Format vihzova programmih: ");
   Cout<<("\n        %s     fayl_istochnik  fayl_priemnik",
              argv[0]);
               return 1;
  }
if  ((fdin =open(argv[1],O_RDONLY)) ==-1)
{
   perror (argv[1]);
  return 1;
}
if ((fdout=open(argv[2],
       O_WRONLY|O_CREAT|O_TRUNC))== -1)
{
perror (argv[2]);
return 1;
}
/* faylih otkrihtih – mojno  kopirovat   */
while  ((n=read(fdin, buff, BUFSIZ))>0)
   write (fdout, buff,  n );
return 0;
} /* konets programma */
```

BUFSIZ konstantasi iostream.h sarlavhali faylda aniqlangan bo'lib MS DOS uchun 512 bayt ga teng

57 - DARS. FAYLGA IHTIYORIY MUROJAAT.

Quyida darajada fayllarni ihtiyoriy tartibda uqish mumkin. Buning uchun lseek ( ) funktsiyasidan foydalanish lozim. Bu funktsiya prototipi quyidagi ko'rinishga ega:

Long lseek (int fd, long offset, int origin);
Bu funktsiya fd deskriptori bilan bog'lik fayldagi joriy pozitsiyani uchinchi parametr (origen) orqali nuqtaga nisbatan ikkinchi parametr (offset) qadamga ko'taradi. Boshlangich nuqta MS DOS da io.h yoki UNIX da unistd.h sarlavhali fayllarda aniqlangan konstantalar orqali aniqlanadi:

SEEK_SET (0 qiymatga ega) fayl boshi

SEEK_CUR (1 qiymatga ega) joriy pozitsiya

SEEK_END (2 qiymatga ega) fayl ohiri

Ko'chish davomida hato kelib chiqsa hato kodi errno global o'zgaruvchisiga yoziladi.
Faylda joriy pozitsiyani aniqlash uchun tell ( ) funktsiyasidan foydalaniladi:
Bu funktsiya prototipi : long tell (int fd) ;
Joriy pozitsiyani fayl boshiga keltirish:
Lseek (fd, oh, seek_set)
Joriy pozitsiyani fayl ohiriga keltirish:
Lseek (fd, oh, seek_end)

58 - DARS. STRUKTURALI TIPLAR VA STRUKTURALAR.

Strukturali tip.

Struktura bu turli tipdagi ma'lumotlarning birlashtirilgan tipdir. Struktura har hil tipdagi elementlar-komponentalardan iborat buladi. Strukturalar qo'yidagicha ta'riflanishi mumkin:

Struct struturali_tip_nomi
{Elementlar_ta'riflari}

Misol uchun ombordagi mollarni tasvirlovchi strukturani quramiz. Bu struktura qo'yidagi komponentalarga ega bo'lishi mumkin:

- Mol nomi (char*)
- Sotib olish narhi (long)
- Ustiga quyilgan narh, foizda (float)
- Mol soni (int)
- Mol kelib tushgan sana (char[9])

Bu struktura dasturda qo'yidagicha ta'riflanadi:

```
 struct goods {
char* name;
long price;
float percent;
int vol;
char date[9];
} year;
```

Konkret strukturalar va strukturaga ko'rsatkichlar bu tip yordamida qo'yidagicha ta'riflanishi mumkin:

Struct goods food, percon; struct goods *point_to;

Strukturalarni tasvirlashda ihtiyoriy murakkab tip uchun nom berishga imkon beruvchi typedef hizmatchi so'zidan foydalanish mumkin. Bu holda strukturali tip qo'yidagi shaklda ta'riflanadi:

Typedef struct
{Elementlar_ta'riflari}
strukturali_tip_nomi

Misol uchun:

```
Typedef struct
{ double real;
   double imag;
}
complex;
```

Bu misolda kompleks sonni tasvirlovchi strukturali tip complex kiritilgan bo'lib, kompleks son haqiqiy qismini tasvirlovchi real va mavhum qismini tasvirlovchi komponentalaridan iboratdir. Konkret strukturalar bu holda qo'yidagicha tasvirlanadi:

Complex sigma,alfa;

Strukturali tip typedef yordamida aniqlangan nomdan tashqari, standart usulda aniqlangan nomga ega bo'lishi mumkin. Qo'yidagi misolda kasr sonni tasvirlovchi numerator –sur'at va denominator-mahraj komponentalaridan iborat struktura ta'rifi keltirilgan.

```
Typedef struct  rational_fraction
{ int numerator;
  int denominator;
```

} fraction;
Bu misolda fraction kasrning Typedef orqali kiritilgan nomi, rational_fraction standart usulda kiritilgan nom. Bu holda konkret strukturalar qo’yidagicha tasvirlanishi mumkin:
Struct rational_fraction alfa; fraction beta;

59 - DARS. KONKRET STRUKTURALARNI TASVIRLASH.

Yuqoridagi misollarda konkret strukturalarni ta'riflashni ikki usuli ko’rib chiqilgan. Agar strukturali tip standart usulda kiritilgan bo’lsa konkret strukturalar qo’yidagi shaklda ta'riflanadi:

Struct < struktura nomi> <konkret strukturalar ruyhati>

Masalan Struct goods food

Agar strukturali tip typedef hizmatchi so’zi yordamida kiritilgan bo’lsa konkret strukturalar qo’yidagi shaklda ta'riflanadi:

< struktura nomi> <konkret strukturalar ruyhati>

Masalan Complex sigma

Bu usullardan tashqari konkret strukturalarni ta'riflashning boshqa usullari ham mavjuddir. Strukturalar ta'riflanganda konkret strukturalar ruyhatini kiritish mumkin:

Struct struturali_tip_nomi
{Elementlar_ta'riflari}
Konkret_strukturalar_ruyhati.

Misol:

```
Struct student
{
char name[15];
char surname[20];
int year;
} student_1, student_2, student_3;
```

Bu holda student strukturali tip bilan birga uchta konkret struktura kiritiladi. Bu strukturalar student ismi (name[15]), familiyasi (surname[20]), tugilgan yilidan (year) iborat.

Strukturali tip ta'riflanganda tip nomi ko’rsatilmay, konkret st'rukturalar ruyhati ko’rsatilishi mumkin:

```
Struct
{Elementlar_ta'riflari}
Konkret_strukturalar_ruyhati.
```

Qo’yidagi ta'rif yordamida uchta konkret struktura kiritiladi, lekin strukturali tip kiritilmaydi.

```
struct {
        char processor [10];
        int frequency;
int memory;
int disk;
} IBM_486, IBM_386, Compaq;
```

60 - DARS. STRUKTURALAR UCHUN HOTIRADAN JOY AJRATISH.

Strukturali tip kiritilishi bu tip uchun hotiradan joy ajratilishiga olib kelmaydi. Har bir konkret struktura (ob'ekt) ta'riflanganda, shu ob'ekt uchun elementlar tiplariga qarab hotiradan joy ajratiladi. Hotiradan joy zich ajratilganda struktura uchun ajratilgan joy hajmi har bir element uchun zarur bo'lgan hotira hajmlari yig'indisiga teng bo'ladi. Shu bilan birga hotiradan joy zich ajratilmasligi ham mumkin ya'ni elementlar orasida bo'sh joylar ham qolishi mumkin. Bu bo'sh joy keyingi elementni hotira qismlarining qabul qilingan chegaralari bo'yicha tekislash uchun qoldiriladi. Misol uchun butun tipdagi elementlar juft adreslardan boshlansa, bu elementlarga murojaat tezroq amalga oshiriladi. Konkret strukturalarni joylashuviga ba'zi kompilyatorlarda #pragma preprotsessor direktivasi yordamida ta'sir o'tkazish mumkin. Bu direktivadan qo'yidagi shaklda:

Pragma pack(n)

Bu erda n qiymati 1,2 eki 4 ga teng bo'lishi mumkin.

Pack(1) – elementlarni bayt chegaralari bo'yicha tekislash;

Pack(2) – elementlarni so'zlar chegaralariga qarab tekislash;

Pack(4) – elementlarni ikkilangan muzlar chegaralariga qarab tekislash.

Struktura uchun ajratilgan joy hajmini qo'yidagi amallar yordamida aniqlash mumkin:

Sizeof (strukturali_tip_nomi);

Sizeof (struktura_nomi);

Sizeof struktura_nomi.

Ohirgi holda struktura nomi ifoda deb qaraladi. Ifodaning tipi aniqlanib, hajmi hisoblanadi.

Misol uchun:

Sizeof (struct goods)

Sizeof (tea)

Sizeof coat

61 - DARS. STRUKTURALARGA MUROJAAT.

Konkret strukturalar ta'riflanganda massivlar kabi initsializatsiya qilinishi mumkin.
Masalan

```
complex sigma {1.3;12.6};
Struct goods coats={“pidjak’,40000,7.5,220,”12.01.97”);
```

Bir hil tipdagi strukturalarga kiymat berish amalini kullash mumkin:

```
Complex alfa; alfa=sigma;
```

Lekin strukturalar uchun solishtirish amallari aniqlanmagan.
Strukturalar elementlariga qo’yidagicha murojaat qilish mumkin:
Struktura nomi.element_nomi.
Nuqta amali’ struktura elementiga murojaat qilish amali deyiladi. Bu amal qavs amallari bilan birga eng yuqori ustivorlikka egadir.
Misol:

```
Complex alfa={1.2,-4.5},betta={5.6,-7.8),sigma;
Sigma.real=alfa.real+betta.real;
Sigma.imag=alfa.imag+betta.imag;
```

Konkret strukturalar elementlari dasturda alohida kiritilishi va chiqarilishi zarurdir.
Qo’yidagi misolda ikki kompleks son qiymatlari kiritilib, yigindisi hosil qilinadi:

```
 #include  <iostream.h>
typedef struct {
double real;
double imag;
} complex;
void main()
{
complex x,y,z;
Cout<<(“\n                :”);Cin>>(“%f”,&x.real);
Cout<<(“\n                :”);Cin>>(“%f”,&x.imag);
Cout<<(“\n                :”);Cin>>(“%f”,&y.real);
Cout<<(“\n                :”);Cin>>(“%f”,&y.imag);
z.real=x.real+y.real;
z.imag=x.imag+y.imag;
Cout<<(“\n            %f”,&z.real);
Cout<<(“\n            %f”,&z.imag);
}
```

62 - DARS. STRUKTURALAR VA MASSIVLAR.

Massivlar strukturalar elementlari sifatida.
Massivlarni strukturalar elementi sifatida ishlatilishi hech qanday qiyinchilik tug'dirmaydi. Biz yuqorida simvolli massivlardan foydalanishni ko'rdik. Qo'yidagi misolda fazoda berilgan nuqtaviy jismni tasvirlovchi komponentalari jism massasi va koordinatalaridan iborat struktura kiritilgan bo'lib, nuqtaning koordinatalar markazigacha bo'lgan masofasi hisoblangan.

```
 Include <iostream.h>
#include <math.h>
void main()
{
struct
 {
double mass;
float coord[3]
} point={12.3,{1.0,2.0,-3.0}};
int i;
float s=0.0;
for (i=0;i<3; i++)
s+=point.coord[i]*point.coord[i];
Cout<<("\n masofa=%f",sqrt(s));
}
```

Bu misolda point strukturasi nomsiz strukturali tip orqali aniqlangan bo'lib, qiymatlari initsializatsiya yordamida aniqlanadi.

Strukturalar massivlari.

Strukturalar massivlari oddiy masivlar kabi tasvirlanadi. Yuqorida kiritilgan strukturali tiplar asosida qo'yidagi strukturalar massivlarini kiritish mumkin:

```
Struct goods list[100];
Complex set [80];
```

Bu ta'riflarda list va set strukturalar nomlari emas, elemenlari strukturalardan iborat massivlar nomlaridir. Konkret strukturalar nomlari bo'lib set[0],set[1] va hokazolar hizmat qiladi. Konkret strukturalar elementlariga qo'yidagicha murojaat qilinadi:
set[0].real– set massivi birinchi elementining real nomli komponentasiga murojaat.
Quyidagi misolda nuktaviy jismlarni tasvirlovchi strukturalar massivi kiritiladi va bu nuqtalar sistemasi uchun og'irlik markazi koordinatalari (xc,yc,zc) hisoblanadi. Bu koordinatalar qo'yidagi formulalar asosida hisoblanadi:
m=?mi; xc =(?ximi)/m; yc =(?yimi)/m; zc =(?zimi)/m;

```
#Include <iostream.h>
struct particle {
double mass;
float coord [3];
};

struct particle mass_point[]={ 20.0, {2.0,4.0,6.0}
                               40.0, {6.0,-2.0,6.0}
                               10.0, {1.0,3.0,2.0}
```

```
                                                    };
int N;
struct particle center ={ 0.0, {0.0,0.0,0.0}
                                                  }
int I;
N=sizeof(mass_point)/sizeof(mass_point[0]);
For (I=0;I<N;I++)
{
center.mass+=mass_point[I].mass
center.coord[0]+=  mass_point[I].coord[0]* mass_point[I].mass;
center.coord[1]+=  mass_point[I].coord[1]* mass_point[I].mass;
center.coord[2]+=  mass_point[I].coord[2]* mass_point[I].mass;
}
Cout<<("\n Koordinatih tsentra mass:");
for (I=0;I<3;I++)
{
center.coord[I]/=center.mass;
Cout<<("\n Koordinata %d:%f",(I+1),center.coord[I]);
}
}
```

63 - DARS. STRUKTURALAR VA KO'RSATKICHLAR.

Strukturaga ko'rsatkichlar oddiy ko'rsatkichlar kabi tasvirlanadi:

```
Complex *cc,*ss; struct goods *p_goods;
```

Strukturaga ko'rsatkich ta'riflanganda initsializatsiya qilinishi mumkin. Misol uchun ekrandagi rangli nuktani tasvirlovchi qo'yidagi strukturali tip va strukturalar massivi kiritiladi. Strukturaga ko'rsatkich qiymatlari initsializatsiya va qiymat berish orqali aniqlanadi:

```
Struct point
{int color;
  int  x,y;
} a,b;
 struct point *pa=&a,pb; pb=&b;
```

Ko'rsatkich orqali struktura elementlariga ikki usulda murojaat qilish mumkin. Birinchi usul adres bo'yicha qiymat olish amaliga asoslangan bo'lib qo'yidagi shaklda qo'llaniladi:

```
(* strukturaga ko'rsatkich).element nomi;
```

Ikkinchi usul mahsus strelka (->) amaliga asoslangan bo'lib qo'yidagi ko'rinishga ega:

```
 strukturaga ko'rsatkich->element nomi
```

Struktura elementlariga qo'yidagi murojaatlar o'zaro tengdir:

```
(*pa).color==a.color==pa->color
```

Struktura elementlari qiymatlarini ko'rsatkichlar yordamida qo'yidagicha o'zgartirish mumkin:

```
(*pa).color=red;
pa->x=125;
pa->y=300;
```

Dasturda nuqtaviy jismni tasvirlovchi particle strukturali tipga tegishli m_point strukturasi aniqlangan bo'lsin. Shu strukturaga pinta ko'rsatkichini kiritamiz:

```
Struct particle * pinta=&m_point;
```

Bu holda m_point struktura elementlarini qo'yidagicha o'zgartirish mumkin:

```
Pinta->mass=18.4;
For (I=0;I<3;I++)
Pinta->coord[I]=0.1*I;
```

Strukturalarga ko’rsatkichlar ustida amallar.

Strukturalarga ko’rsatkichlar ustida amallar oddiy ko’rsatkichlar ustida amallardan farq qilmaydi. Agar ko’rsatkichga strukturalar massivining biror elementi adresi qiymat sifatida berilsa, massiv buyicha uzluksiz siljish mumkin buladi. Misol tariqasida kompleks sonlar massivi summasini hisoblash masalasini ko’rib chiqamiz:

```
#include <iostream.h>
void main()
{
sruct complex
 {float x;
   float y;} array[]={1.0,2.0,3.0,-4.0,-5.0,-6.0,-7.0,-8.0};
struct complex summa={0.0,0.0};
struct complex *point=&array[0];
int k,I;
k=sizeof(array)/sizeof(array[0]);
for(i=0;i<k;i++)
{
summa.x+=point->x;
summa.y+=point->y;
point++;
}
Cout<<(“\n Summa: real=%f”,\t imag=%f”,summa.x,summa.y);
}
```

Dastur bajarilishi natijasi:
Summa: real=-8.000000, imag=-16.000000

Strukturalar va funktsiyalar.

Strukturalar funktsiyalar argumentlari sifatida yoki funktsiya qaytaruvchi qiymat kelishi mumkin. Bundan tashqari ikkala holda ham strukturaga ko’rsatkichlardan foydalanish mumkindir. Misol uchun kompleks son modulini hisoblash dasturini keltiramiz:

```
Double modul(complex a)
{return sqrt(a.real*a.real+a.imag*a.imag}
```

Ikki kompleks son yigindisini hisoblash funktsiyasi:

```
Complex add(complex a, complex b)
{ complex c;
c.real=a.real+b.real;
c.imag=a.imag+b.imag;
return c;
}
```

Bu funktsiyani ko’rsatkichlar yordamida qo’yidagicha yozish mumkin

```
Complex* add(complex* a, complex* b)
{ complex* c; c=(complex*)malloc(sizeof(complex));
c->real=(*a).real+(*b).real;
c->imag=(*a).imag+(*b).imag;
return c;
```

```
}
```
Bu funktsiya complex tipidagi dinamik ob'ekt yaratib adresini qaytaradi. Dasturda bu ob'ekt uchun ajratilgan joyni ozod qilish maqsadga muvofiq. Bu funktsiyaga dasturda qo’yidagicha murojaat qilish mumkin:

```
Complex a={0.1,-0.3},b={0.2,-0.5};
Complex* pa; pa=add(&a,&b);
```

64 - DARS. ABSTRAKT TIPLARNI TASVIRLASH.

Amaliy masalalarni echishda, shu soha uchun mos bo'lgan ma'lumotlar tiplarini aniqlab olish qo'laydir. Dasturda bu tiplar strukturali tiplar sifatida tasvirlanadi. Sungra shu tip bilan bog'lik hamma funktsiyalarni ta'riflab, biblioteka hosil qilinadi. Misol tariqasida kasrlar bilan bog'lik abstrakt tip kiritamiz. Kasrlar ustida quyidagi funktsiyalarni kiritamiz.

Input() kasr soni kiritish;
Out() kasr soni ekranga chiqarish;
Add() kasrlarni qo'shish;
Sub() kasrlarni ayirish;
Mult() kasrlarni kupaytirish;
Divide() kasrlarni bo'lish;

```
//* fract.h
Typedef struct rational_fraction
{
        int numerator;
        int denominator;
        } fraction;
void input (fraction * pd);
void out( fraction dr);
fraction add (fraction dr1, fraction dr2);
void sub (fraction dr1, fraction dr2, fraction * prd);
fraction * mult ( fraction dr1, fraction dr2);
fraction divide ( fraction * pd1, fraction * pd2);

//* fract.c
#include <iostream.h>
#include <stdlib.h>
void input (fraction * pd)
{
        int N;
Cout<<("\n Chislitel':");
Cin>>("%d", pd->numarator);
Cout<<("Znamenatel':");
Cin>>("%d", &N);
if (N==0)
{
        Cout<<("\n" Oshibka! Nulevoy znamenatel'");
exit (0);
}
pd-> denominator=N;

void out  ( fraction dr)
{
        Cout<<(" Ratsional'naya drob'");
        Cout<<( "%d/%d", dr.numerator, dr.denominator);
}
```

```
fraction add ( fraction dr1, fraction dr2)
{
        fraction dr;
dr.numerator=dr1.numarator * dr2.denominarator+ dr1.denominator * dr2.numarator;

dr.denominator=dr1.denominator * dr2.denominator;

return dr;
}

void sub ( fraction dr1, fraction dr2, fraction * pdr)
{
pdr-> numarator=dr1.numarator * dr2.numarator- dr2.numarator *
dr1.denominator;

pdrw-> denominator= dr1.denominator* dr2.denominator;
        }

fraction * mult ( fraction dr1, fraction dr2 )
{
        fraction * mul;
mul= (fraction 8) malloc (sizeof ( fraction) );
mul-> nymerator=dr1.numerator * dr2.numerator;
mul-> denominator= dr1.denominator * dr2.denominator;
return 0;
}

fraction divide ( fraction * pd1,  fraction * pd2)
{
        fraction d;
d.numarator= pd1-> numarator * pd2 ->denominator;
d.denominator=pd1->denominator * pd2 ->numarator;
return d;
}
```

Qo'yidagi programma kiritilgan funktsiyalar yordamida kasrlar ishlashga misol bo'ladi

```
#Include "fract.h"
#include "fract.c"
void main()
{
fraction a,b,c;
fraction *p;
Cout<<("\n Vvedite drobi");
input(&a);
input(&b);
c=add(a,b);
out©;
p=mult(a,b);
out(*p);
free(p);
```

```
c=divide(&a,&b);
out(c);
}
```

65 - DARS. BIRLASHMALAR.

Strukturalarga yaqin tushuncha bu birlashma tushunchasidir. Birlashmalar union hizmatchi so'zi yordamida kiritiladi. Misol uchun union {long h; int I,j; char c[4]}UNI; Birlashmalarning asosiy hususiyat shundaki uning hamma elementlari bir hil boshlangich adresga ega bo'ladi.

Qo'yidagi dastur yordamida bu hususiyatni tekshirish mumkin:

```
#include <iostream.h>
void main()
{  union {long h; int k; char c[3]}U={10l;-3;"ALI");
Cout<<("\n l=%d";&u.l);
Cout<<("\n k=%d";&u.k);
Cout<<("\n c=%d";&u.c);
};
```

Birlashmalarning asosiy avfzalliklaridan biri hotira biror qismi qiymatini har hil tipdagi qiymat shaklida qarash mumkindir. Misol uchun qo'yidagicha birlashma union {float f; unsigned long k; char h[4];}fl;

Hotiraga fl.f=2.718 haqiqiy son yuborsak uning ichki ko'rinishi kodini fl.l yordamida ko'rishimiz, yoki alohida baytlardagi qiymatlarni fl.h[0]; fl.h[1] va hokazo yordamida qo'rishimiz mumkin.

Birlashmalar imkoniyatlarini ko'rsatish uchun bioskey() funktsiyasidan foydalanishni ko'rib chiqamiz. Bu funktsiya bios.h sarlavhali faylda joylashgan bo'lib, qo'yidagi prototipga ega:

```
int bioskey(int);
```

MS DOS operatsion tizimida ihtiyoriy klavishaning bosilishi klaviatura buferiga ieei bayt ma'lumot yozilishiga olib keladi.

Agar funktsiyaga bioskey(0) shaklda murojat qilinsa va bufer bo'sh bo'lsa biror klavishaga bosilishi kutiladi, agar bufer bo'sh bo'lmasa funktsiya buferdan ikki baytli kodli o'qib butun son sifatida qaytaradi. Funktsiyaga bioskey(0) shaklda murojat qilinsa va bufer bo'sh bo'lsa biror klavisha bosilishi kutiladi, agar bufer bo'sh bo'lmasa funktsiya buferdagi navbatdagi kodni qaytaradi. Funktsiyaga bioskey(1) shaklda murojat qilish bufer bush yoki bo'shmasligini aniqlashga imkon beradi. Agar bufer bo'sh bo'lmasa funktsiya buferdagi navbatdagi kodni qaytaradi, lekin bu kod buferdan o'chirilmaydi. Qo'yidagi dastur buferga kelib tushuvchi kodlarni ukib monitorga chikarishga imkon beradi:

```
# include <iostream.h>
# include <bios.h>
void main()
{
union
{ char rr[2];
int ii;
} cc;
unsigned char scn,asc;
Cout<<("\n\n Ctrl+Z bilan chikish."
Cout<<("\n Klavigani bosib, kodini oling. \n ");
Cout<<("\n    SCAN   ||   ASCII");
Cout<<("\n  (10) (16)    (10) (16)");
do
{ Cout<<("\n");
```

```
cc.ii=bioskey(0);
asc=cc.hh[0];
scn=cc.hh[1];
Cout<<( “  %4d  %3xH || %4d %3xH |”,scn,scn,asc,asc);
}
while(asc!=26 || scn!=44);
}
```

Bu dasturda cc nomli birlashma kiritilgan bo’lib, cc.ii elementiga bioskey(0) funktsiyasi natijasi yoziladi. So’ngra natijaning alohida baytlari sken va ASCII kodlar sifatida monitorga chiqariladi.

Tsikl to 26 ASCII kod va 44 sken kod paydo bo’lmaguncha (CTRL+Z klavishlari bosilmaguncha) davom etadi.

66 - DARS. DINAMIK INFORMATSION STRUKTURALAR.

Ma'lumotlarni statik va dinamik tasvirlash.

Dasturlash tillarida o'zgaruvchilar tiplari va ta'rifi bu o'zgaruvchilar uchun ajratiladigan hotira turini va o'zgaruvchilar qiymatlari chegaralarini belgilaydi. Asosiy tiplar (int, double va hokazo) lar uchun bu belgilar kompilyatorga bog'likdir. Murakkab tiplar ya'ni massivlar va strukturali tiplar uchun hotiraga talab ularning ta'rifiga bog'likdir. Masalan double array[10] ta'rif hotiradan 10*sizeof bayt joy ajratilishiga olib keladi.

```
Struct mixture
{
int ii;
long ll;
char cc[8];
};
```

Bu ta'rif har bir Struct mixture tipidagi ob'ekt hotirada sizeof(int)+sizeof(long)+8*sizeof(char) bayt joy egallashini ko'rsatadi. Ob'ekt aniq hajmini qo'yidagi amal hisoblaydi:

```
Sizeof(struct mixture)
```

Bu usulda kiritilayotgan ob'ektlar faqat statik ma'lumotlarni tasvirlashga imkon beradi. Lekin ko'p misollarda tarkibi, hajmi o'zgaruvchan bo'lgan murakkab konstruktsiyalardan foydalanishga to'gri keladi. Bunday o'zgaruvchan ma'lumotlar dinamik informatsion strukturalar deb ataladi.

Bir elementli ruyhat.

Eng soda dinamik informatsion struktura elementlari qo'yidagi strukturali tip orqali ta'riflangan ob'ektlardan iborat ruyhatdir.

```
Struct strukturali tip nomi
{
struktura elementlari ;
Struct strukturali tip nomi*kursatkich;
};
```

Qo'yidagi misolni ko'rib chiqamiz, Klaviatura orqali ihtiyoriy sondagi strukturalarni bir bog'lamli ro'yhatga birlashgan holda kiritish, so'ngra ruyhat elementlarini kiritilgan tartibda ekranga chiqarish. Ruyhat bilan ishlash uchun uchta ko'rsatkichdan foydalaniladi: beg ruyhat boshiga ko'rsatkich, end ruyhat ohiriga ko'rsatkich, rex ro'yhatni boshidan qarab chiqish uchun ishlatiladigan ko'rsatkich.

Qo'yidagi dastur qo'yilgan vazifani bajaradi:

```
#Include <stdlib>
#include <stdio.h>
struct cell {
              char sign[10];
              int weight;
              struct cell * pc;
             };
void main()
{
  struct cell * rex;
  struct cell * beg=NULL;
  struct cell * end=NULL;
do
```

```
{
    rex=(struct cell*malloc(sizeof(struct cell));
printf(“sign=”);
scanf(“%s, & rex->sign);
printf(“weight=”);
scanf(“%d”,&rex->weight);
if (rex->weight==0)
{
   free(rex);
   break;
}
if (beg==NULL&&end==NULL)
    beg=rex;
else
    end->pc=rex;
end=rex;
end->pc=NULL;
}
while(1);
printf(“\nRuyhatni chikarish:”);
rex=beg;
while(rex!=NULL);
{
   printf(“\nsign=%s\tweight=%d”,rex->sign,rex->weight);
rex=rex->pc;
  }
}
```

Dastur bajarilishiga misol:

Struktura haqidagi ma'lumotni kiriting:

Sign=sigma
Weight=16
Sign=omega
Weight=44
Sign=alfa
Weight=0

Ruyhatni chiqarish

Sign=sigma weight=16
Sign=omega weight=44

Dasturda ma'lumotlarni kiritish tsikl orqali bajariladi. Tsikl tugatilishi sharti navbatdagi strukturaning `int weight` elementiga kiritilgan nol qiymatdir.
Strukturalar ro’yhati dinamik tashkil etiladi. Beg va end ko’rsatkichlar nol qiymati orqali initsializatsiya qilingan. Dasturda `(struct cell*)` tip o’zgartirishidan foydalanilgan chunki `malloc` funktsiyasi har doim void tipidagi ko’rsatkich qaytaradi.

66 - DARS. RUYHATLAR BILAN ISHLASHDA REKURSIYADAN FOYDALANISH.

Ruyhatning har bir buginida ma'lumot va keyingi element adresi joylashgan.Agar bu ko’rsatkich nol qiymatga ega bo’lsa ro’yhat ohirigacha o’qib bo’lingan. Ruyhatni kurib chiqishni boshlash uchun birinchi elementining adresini bilish etarlidir. Ruyhatni yaratish rekursiv funktsiyasi va ruyhatning elementlarini ekranga chiqarish rekursiv funktsiyasini kurib chiqamiz. Ruyhatni rekursiv tuldirish funktsiyasi quyidagi prototipga ega:

```
Struct cell* input(void);
```

Bu funktsiya klaviatura orqali kiritilgan ma'lumotlar bilan to’ldirilgan ro’yhatga ko’rsatkich qaytaradi. Funktsiyaga har gal murojaat qilinganda yangi ruyhat yaratiladi. Agar ruyhatning navbatdagi strukturasining weight o’zgaruvchisiga nol qiymat berilsa funktsiya o’z ishini to’htatadi. Aks holda klaviatura orqali tuldirilgan struktura ruyhatga ulanadi va uning struct cell*pc elementining qiymati funktsiya tanasidan rekursiv chaqirilgan input() funktsiyasi qaytargan qiymatga teng bo’ladi.

Funktsiya tanasi:

```
Struct cell
 {
   char sign[10];
   int weight;
   struct cell*pc;
};
#include <stdlib.h>
#include <stdio.h>
struct cell * input(void)
{
  struct cell*p;
  p=(struct cell*) malloc(sizeof(struct sell));
  printf(“sign=”);
  scanf(“%s”,&p->weight);
  if (p->weight==0)
  {
    free(p);
    return NULL;
   }
p->pc=input();
   return p;
/* */

void output(struct cell * p)
{
   if (p==NULL)
   {
      printf(“\nRuyhat tugadi”);
      return;
    }
printf(“\nsign=%s\tweight=%d”,p->sign,p->weight);
output(p->pc);
}
```

```
void main()
{
  struct cell * beg=NULL;
    printf("\Ruyhat elementlarini kiriting:\n");
beg=input();
printf("\n Ruyhatni chikarish:");
output(beg);
}
```

Dastur bajarilishiga misol:

Ruyhat elementlarini kiriting:
Sign=Zoro
Weight=1938
Sign=Zodiac
Weight=1812
Sign=0
Weight=0

Ro'yhatni chiqarish:
Sign=Zoro weight=1938
Sign=Zodiac weight=1812
Ruyhat tugadi

67 - DARS. C++ DA SINFLAR.
SINF-STRUKTURA TUSHUNCHASI KENGAYTMASI SIFATIDA.

Sinflarni eng soda holda qo'yidagicha tasvirlash mumkin:
Sinf-kaliti Sinf-soni {komponentalar ruyhati}
Sinf komponentalari sodda holda tiplangan ma'lumotlar va funktsiyalardan iborat bo'ladi. Figurali qavslarga olingan komponentalar ro'yhati sinf tanasi deb ataladi. Sinfga tegishli funktsiyalar komponenta-funktsiyalar yoki sinf funktsiyalari deb ataladi. Sinf kaliti sifatida Struct hizmatchi so'zi ishlatilishi mumkin. Masalan qo'yidagi konstruktsiya kompleks son sinfini kiritadi.

```
Struct complex 1
{ double real;
   double imag;
   void define (double re=0.0, double im=0.0)
{ real=re; imag=im;}
   void display (void)
{cout<=”real=”<<real;
  cout<=”imag=”<<imag;
}
};
```

Strukturadan bu sinfning farqi shuki komponenta ma'lumotlardan (real, imag) tashqari ikkita komponenta funktsiya (define() va display ()) kiritilgan. Bu kiritilgan sinf o'zgaruvchilar tipi deb qaralishi mumkin. Bu tiplar yordamida konkret ob'ektlarni qo'yidagicha tasvirlash mumkin:
Misol uchun:

```
Complex x,y;
Complex dim[8];
Complex *p=1x;
```

Sinfga tegishli ob'ektlar qo'yidagicha tasvirlanadi;
   Sinf-nomi . ob'ekt-nomi
Dasturda ob'ekt komponentasiga quyidagicha murojaat qilish mumkin:
 Sinf-nomi.ob'ekt-nomi :: komponenta-nomi yoki soddaroq holda ob'ekt-nomi. Element-nomi
Misol uchun:

```
x!=real=1.24;
x!=imag=0.0;
dim[3]. Real=0.25;
dim[3]. Imag=0.0;
```

Sinfga tegishli funktsiyalarga qo'yidagicha murojaat qilinadi:
funktsiya-nomi.ob'ekt-nomi;
Misol uchun:
X. define.(Bu holda real=0.9 va imag=0.0)
X. define.(Bu holda kompleks son 4.3+i*20.0)
Display funktsiyasi ekranda kompleks son qiymatlarini tasvirlaydi. Sinfga tegishli ob'ektga ko'rsatkich orqali komponentalarga quyidagicha murojat qilinadi:
Ob'ektga-ko'rsatkich>element-nomi
Yuqorida ko'rsatilgan P ko'rsatkich orqali H ob'ekt elementlariga qo'yidagicha qiymat berish mumkin:
P>real=2.3

P>imag=6.1
Huddi shu shaklda sinfga tegishli funktsiyalarga murojat qilinadi:
P>display;
P>define(2.3, 5.4);

Kompanenta o’zgaruvchilar va kompanenta funktsiyalar.

Sinf kompanenta o’zgaruvchilari sifatida o’zgaruvchilar , massivlar, ko’rsatkichlar ishlatilishi mumkin . Elementlar ta'riflanganda initsializatsiya qilish mumkin emas . Buning sababi shuki sinf uchun hotiradan joy ajratilmaydi. Kompanenta elementlariga kompanenta funktsiyalar orqali murojat qilinganda faqat nomlari ishlatiladi. Sinfdan tashqarida sinf elementlariga emas ob'ekt elementlariga murojaat qilish mumkin. Bu murojaat ikki hil bo’lishi mumkindir.
Ob'ekt- nomi . Element - nomi.
Ob'ktga – korsatgich – element nomi.
Sinf elementlari sinfga tegishli funktsiyalarida ishlatilishidan oldin ta'riflangan bo’lishi shart emas. Huddi shunday bir funktsiyadan hali ta'rifi berilmagan ikkinchi funktsiyaga murojaat qilish mumkin. Komponentalarga murojaat huquqlari.
Komponentalarga murojaat huquqi murojaat spetsifikatorlari yordamida boshqariladi.
Bu spetsifikatorlar :
Protected – himoyalangan;
Private – hususiy;
Public – umumiy;
Himoyalangan kompanentalardan sinflar ierarhiyasi qurilganda foydalaniladi. Oddiy holda Protected spetsifikatori Private spetsifikatoriga ekvivalentdir. Umumiy ya'ni Public tipidagi komponentalarga dasturning ihtiyoriy joyida murojaat qilinishi mumkin. Hususiy ya'ni Private tipidagi komponentalarga sinf tashqarisidan murojaat qilish mumkin emas.
Agar sinflar Struct hizmatchi so’zi bilan kiritilgan bo’lsa, uning hamma komponentalari umumiy Public bo’ladi, lekin bu huquqni murojaat spetsifikatorlari yordamida o’zgartirish mumkin. Agar sinf Class hizmatchi so’zi orqali ta'riflangan bo’lsa, uning hamma komponentalari hususiy bo’ladi. Lekin bu huquqni murojaat spetsifikatorlari yordamida uzgartirish mumkindir. Bu spetsifikator yordamida Sinflar umumiy holda quyidagicha ta'riflanadi:

```
class class_name
{
  int data_member; // Ma'lumot-element
  void show_member(int); // Funktsiya-element
};
```

Sinf ta'riflangandan so’ng, shu sinf tipidagi o’zgaruvchilarni(ob'ektlarni) qo’yidagicha ta'riflash mumkin:

```
class_name object_one, object_two, object_three;
```

Qo’yidagi misolda employee, sinfi kiritilgandir:

```
class employee
{
  public:
  char name[64] ;
  long employee_id;
  float salary;
  void show_employee(void)
  {
    cout << "Imya: " << name << endl;
    cout << "Nomer slujathego: " << employee_id << endl;
```

```
    cout << "Oklad: " << salary << endl;
  };
};
```

Bu sinf uch o’zgaruvchi va bitta funktsiya-elementga ega. Qo’yidagi EMPCLASS.CPP dastur ikki employee ob'ektini yaratadi. Nuqta operatordan foydalanib ma'lumot elementlarga qiymat beriladi so’ngra show_employee elementidapn foydalanib hizmatchi haqidagi ma'lumot ekranga chiqariladi:

```
#include <iostream.h>
#include <string.h>
class employee
{
public:
  char name [64];
  long employee_id;
  float salary;
  void show_employee(void)
  {
    cout << "Imya: " << name << endl;
    cout << "Nomer slujathego: " << employee_id << endl;
    cout << "Oklad: " << salary << endl;
  };
};
void main(void)
{
  employee worker, boss;
  strcpy(worker.name, "John Doe");
  worker.employee_id = 12345;
  worker.salary = 25000;
  strcpy(boss.name, "Happy Jamsa");
  boss.employee_id = 101;
  boss.salary = 101101.00;
  worker.show_employee();
  boss.show_employee();
}
```

68 - DARS. SINF KOMPONENTA FUNKTSIYALARI.

Komponenta funktsiya ta'rifi.

Komponenta funktsiya albatta sinf tanasida ta'riflangan bo'lishi lozim. Global funktsiyalardan farqli komponenta funktsiya sinfning hamma komponentalariga murojaat qilishi mumkin. Funktsiyaning faqat prototipi emas to'la ta'rifi sinf tanasida joylashgan bo'lsa, bu funktsiya joylashtiruvchi (inline) funktsiya hisoblanadi. Ma'lumki inline funktsiyalarda tsikllar, kalit bo'yicha o'tish operatori ishlatilishi mumkin emas. Bundan tashqari bunday funktsiyalar rekursiv funktsiya bo'lolmaydi. Bu chegaralarni engish uchun sinf tanasiga faqat funktsiya prototipi joylashtirilib, funktsiyaning to'la ta'rifi sinf tashqarisida dasturga kiruvchi boshqa funktsiyalar bilan birga beriladi. Komponenta funktsiyani sinf tashqarisida ta'riflanganda, qaysi sinfga tegishli ekanligini qo'yidagi shaklda ko'rsatiladi:

Sinf-nomi :: Komponenta funktsiya-nomi

Sinf tanasiga komponenta funktsiya prototipi qo'yidagi shaklda joylashtiriladi:

Tip funktsiya-nomi(formal-parametrlar-ta'rifi)

Sinf tashkarisida funktsiya qo'yidagi shaklda ta'riflanadi:

Tip sinf-nomi :: funktsiya-nomi(formal-parametrlar-spetsifikatsiyasi)

{ funktsiya tanasi };

Oldingi misoldagi employee sinfida funktsiya sinf ichida ta'riflangan. Bunday funktsiya joylanuvchi (inline) funktsiya deb qaraladi. Funktsiyani sinf tashqarisida ta'riflab sinf ichiga funktsiya prototipini joylashtirish mumkin. Sinf ta'rifi bu holda qo'yidagi kurinishda bo'ladi:

```
class employee
{
public:
   char name[64];
   long employee_id;
   float salary;
   void show_employee(void); |————————————> Prototip funktsii
};
```

Har hil funktsiyalar bir hil nomli funktsiyalardan foydalanishi mumkin bo'lgani uchun funktsiya nomi sinf nom va global ruhsat operatori belgisi (::) qo'yilishi lozim.

```
void employee:: show_employee (void) //-------------->Imya klassa
{
   sout << "Imya: " << name << endl; Imya elementa cout << "Nomer slujathego: "
<< employee_id << endl;
   cout << "Oklad: " << salary << endl;
};
```

. Qo'yidagi CLASSFUN.CPP dastur show_employee funktsiyasi ta'rifini sinf tashqarisiga joylashtiradi:

```
#include <iostream.h>
#include <string.h>
class employee
{
public:
   char name [64];
   long employee_id;
   float salary;
   void show_employee(void);
```

```
};
void employee::show_employee(void)
{
  cout << "Imya: " << name << endl;
  cout << "Nomer slujathego: " << employee_id << endl;
  cout << "Oklad: " << salary << endl;
};
void main(void)
{
  employee worker, boss;
  strcpy(worker.name, "John Doe");
  worker.employee_id = 12345;
  worker.salary = 25000;
  strcpy(boss.name, "Happy Jamsa");
  boss.employee_id = 101;
  boss.salary = 101101.00;
  worker.show_employee();
  boss.show_employee();
}
```

Ikkinchi misol:

Bu misolda PEDIGREE.CPP dasturida dog, sinfi kiritiladi. Dasturda sinf funktsiyasi show_breed tashqarisida ta'riflanadi. So'ngra dog tipidagi ikki ob'ekt yaratilib, har biri haqidagi ma'lumot ekranga chiqariladi

```
#include <iostream.h>
#include <string.h>
class dogs
{
public:
  char breed[64];
  int average_weight;
  int average_height;
  void show_dog(void) ;
};
void dogs::show_breed(void)
{
  cout << "Poroda: " << breed << endl;
  cout << "Sredniy ves: " << average_weight << endl;
  cout << "Srednyaya vihsota: " << average_height << endl;
}
void main(void)
{
  dogs happy, matt;
  strcpy(happy.breed, "Dolmatin") ;
  happy.average_weight = 58;
  happy.average_height = 24;
  strcpy(matt.breed, "Kolli");
  matt.average_weight =22;
  matt.average_height = 15;
  happy.show_breed() ;
  matt.show_breed();
```

```
}
```
3 misol
Misol tariqasida nuqta tushunchasini aniqlovchi Point.L fayliga yozib qo'yamiz:
```
# includef Point.h
#define Point1.h
class point {
protected:
int x,y;
public;
Point ( int xi=0, int yi=0);
Int& givex(void);
Int& givey (void);
Void show(void);
Void move(int xn=0, int yn=0);
Private:
Void hid();
}
#endif
```
Kelgusida point sinfini boshqa sinflarga qo'shish mumkin bo'lgani uchun shartli protsessor direktivasi #ifndef POINT.H ishlatilgan. Protsessorli identifikator POINT.H #define POINT1.H direktivasi orqali kiritilgan.
Shuning uchun # include "point.h" direktivasi bir necha marta ishlatilganda ham POINT sinfi ta'rifi teksti faqat bir marta kompilyatsiya qilinayotgan faylda paydo bo'ladi. POINT sinfi komponenta funktsiyalarini qo'yidagicha ta'riflaymiz:
```
#ifndef POINT.CPP
#define POINT1.CPP
#include <graphics.h>
#include "point.h"
Point ::point(int xi=0, int yi=0)
{ x=xi; y=yi;}
int &point::givex(void) {return x;}
int &point::givey(void) {return y;}
void point::show(void)
{putpixel(x,y,get color());}
void point::hide (void)
{ putpixel(x,y,get b color());}
void point::move(int xn=0, int yn=0)
{ hide( );
x=xn; y=yn;
show( );
}
#endif
```
sinf ta'rifida qo'yidagi grafik funktsiyalar ishlatiladi:
```
void putpixel(intx; inty; int color)
```
Ekranda color rangli(x,y) kordinatali nuqtani tasvirlaydi.
```
Int getbcolor(void)
```
Fon rangini qaytaradi
```
Int getcolor(void)
```
Tasvir rangini qaytaradi.
Kiritilgan point sinfi va komponenta funktsiyalari bilan ishlovchi dasturni keltiramiz:

```
#include <graphics.h>
#include <conion.h>
#include “point.epp”
void main()
{ point A(200,50);
point B;
point D(500,200);
int dr=Detect, mod;
initgraph(&dr, &mod, ”c:\borland\bgi”);
```

Ichki joylashgan funktsiyalar

Sinf funktsiyalarini sinf ichida yoki tashqarisida ta'riflash mumkindir. Misol uchun employee sinfida funktsiyalar sinf ichida ta'riflangandir:

```
class employee
{
public:
  employee(char *name, char *position, float salary)
  {
    strcpy(employee::name, name);
    strcpy(employee::position, position);
    employee::salary = salary;
  }
  void show_employee(void)
  {
    cout << "Imya: " << name << endl;
    cout << "Doljnost': " << position << endl;
    cout << "Oklad: $" << salary << endl;
  }
private:
  char name [64];
  char position[64];
  float salary;
};
```

Bu holda funktsiyalar joylashtiriluvchi {inline) deb qaraladi. Funktsiya sinf tashqarisida ta'riflangan bo’lsa ularni inline funktsiya sifatida qarash uchun funktsiya ta'rifida inline so’zi aniq ko’rsatilgan bo’lishi kerak:

```
inline void employee::show_employee(void)
{
  cout << "Imya: " << name << endl;
  cout << "Doljnost': " << position << endl;
  cout << "Oklad: $" << salary << endl;
}
```

69 - DARS. KONSTRUKTOR VA DESTRUKTOR

Konstruktorlar.

Konstruktorlar bu sinf komponenta funktsiyalari bulib ,ob'ektlarni avtomatik initsializatsiya qilish uchun ishlatiladi. Konstruktorlar ko'rinishi qo'yidagicha bo'lishi mumkin :

```
Sinf nomi (formal parametrlar ruyhati)
{konstruktor tanasi}
```

Bu komponenta funktsiya nomi sinf nomi bilan bir hil bulishi lozim. Misol uchun complex sinfi uchun konstruktorni qo'yidagicha kiritish mumkin :

```
Mplex (double re = 0.0; double im = 0.0 )
{real=re; imag=im;}
```

Tovarlar sinfi uchun konstruktorni qo'yidagicha kiritish mumkin.

```
Goods(char* new _ name, float new _ price)
{name= new _ name; price= new _ price; }
```

Konstruktorlarda percent kabi statik elementlarning ham qiymatlarini o'zgartirish mumkindir. Konstruktorlar uchun qaytariluvchi tiplar, hatto void tipi ham ko'rsatilmaydi. Dasturchi tomonidan ko'rsatilmagan holda ham ob'ekt yaratilganda konstruktor avtomatik ravishda chaqiriladi.

Masalan ss ob'ekt

Copmlex cc; shaklida aniqlangan bo'lsa, konstruktor avtomatik chaqirilib real va imag parametrlari avtomatik ravishda 0.0 qiymatlariga ega bo'ladi. Ko'rsatilmagan holda parametrsiz konstruktor va qo'yidagi tipdagi nusha olish konstruktorlari yaratiladi: T :: T (const T&)

Misol uchun

```
Class F
{.....
        public : F(const T&)
        .....
        }
```

Sinfda bir nechta konstruktorlar b'olishi mumkin, lekin ularning faqat bittasida parametrlar qiymatlari oldindan ko'rsatilgan bo'lishi kerak. Konstruktor adresini hisoblash mumkin emas. Konstruktor parametri sifatida uz sinfining nomini ishlatish mumkin emas, lekin bu nomga ko'rsatkichdan foydalanish mumkin. Konstruktorni oddiy komponenta funktsiya sifatida chakirib bo'lmaydi. Konstruktorni ikki hil shaklda chaqirish mumkin :

```
Sinf_nomi ,Ob'ekt_nomi (konstruktor_hakikiy_parametlari)
Sinf_nomi (konstruktor_hakikiy_parametlari)
```

Birinchi shakl ishlatilganda haqiqiy parametrlar ro'yhati bo'sh bo'lmasligi lozim. Bu shakldan yangi ob'ekt ta'riflanganda foydalaniladi:

```
Complex SS(10.3; 0.22)
// real=10.3; SS.imag= 0.22;
Complex EE (2.3)
// EE . real= 2.3;
EE.imag= 0.0;
Complex D()  // hato
```

Konstruktorni ikkinchi shaklda chaqirish nomsiz ob'ekt yaratilishiga olib keladi. Bu nomsiz ob'ektdan ifodalarda foydalanish mumkin.

Misol uchun :

Complex ZZ= complex (4.0;5.0);
Bu ta'rif orqali ZZ ob'ekt yaratilib, unga nomsiz ob'ekt qiymatlari(real= 4.0; imag= 5.0) beriladi; Konstruktorlar yordamida ob'ektlar qiymatlarini initsializatsiya qilish uchun initsializatsiya ro'yhatidan foydalanish mumkin:
Sinf_nomi (parametrlar ro'yhati);
Komponenta_uzgaruvchilar_initsializatsiya ruyhati
{konstruktor tanasi}
Initsializatsiya ruyhatining har bir elementi konkret komponentaga tegishli bo'lib, qo'yidagi ko'rinishga ega:
Komponenta_uzgaruvchi_nomi (ifoda)
Misol:

```
Class AZ
        { int ii ; float ee ; char cc ;
public:
AZ (int in ; float en ; char cn) : ii(5),
EE (ii+en+in) , CC(en) { }
.....
} ;
AZ  A(2,3.0,'d');
AZ  X=AZ (0,2.0,'z');
```

Konstruktor nomi sinf nomi Bilan bir hil bo'lishi lozimdir. Misol uchun siz employee sinfdan foydalansangiz, konstruktor ham employee nomga ega bo'ladi. Agar dasturda konstruktor ta'rifi berilgan bo'lsa ob'ekt yaratilganda avtomatik chaqiriladi. Qo'yidagi CONSTRUC.CPP nomli dasturda employee nomli sinf kiritilgandir:

```
class employee
{
public:
   employee(char *, long, float); //Konstruktor
   void show_employee(void);
   int change_salary(float);
   long get_id(void);
private:
   char name [64];
   long employee_id;
   float salary;
};
```

Konstruktor ta'rifi:

```
employee::employee(char *name, long employee_id, float salary)
{
   strcpy(employee::name, name) ;
   employee::employee_id = employee_id;
   if (salary < 50000.0)
   employee::salary = salary;
   else // Nedopustimihy oklad
   employee::salary = 0.0;
}
```

CONSTRUC.CPP dasturi:

```
#include <iostream.h>
#include <string.h>
class employee
```

```
{
public:
  employee(char *, long, float);
  void show_employee(void);
  int change_salary(float) ;
  long get_id(void);
private:
  char name [64] ;
  long employee_id;
  float salary;
};
employee::employee(char *name, long employee_id, float salary)
{
  strcpy(employee::name, name) ;
  employee::employee_id = employee_id;
  if (salary < 50000.0)
    employee::salary = salary;
  else // Nedopustimihy oklad
    employee::salary = 0.0;
}
void employee::show_employee(void)
{
  cout << "Slujathiy: " << name << endl;
  cout << "Nomer slujathego: " << employee_id << endl;
  cout << "Oklad: " << salary << endl;
}
void main(void)
{
  employee worker("Happy Jamsa", 101, 10101.0);
  worker.show_employee();
}
```

Konstruktrdan foydalanilganda ob'ekt ta'rifilanganda parametr uzatish mumkin:
employee worker("Happy Jamsa", 101, 10101.0);
Agar dasturda employee tipidagi ob'ektlar mavjud bo’lsa har birini qo’yidagicha initsializatsiya qilish mumkin

```
employee worker("Happy Jamsa", 101, 10101.0);
employee secretary("John Doe", 57, 20000.0);
employee manager("Jane Doe", 1022, 30000.0);
```

Konstruktorlar va kuzda tutilgan qiymatlar

Konstruktorlarda kuzda tutilgan qiymatlardan ham foydalanish mumkindir. Misol uchun qo’yidagi konstruktor employee oklad qiymatini dasturda ko’rsatilmagan bo’lsa 10000.0 teng qilib oladi.:

```
employee::employee(char *name, long employee_id, float salary = 10000.00)
{
  strcpy(employee::name, name);
  employee::employee_id = employee_id;
  if (salary < 50000.0)
    employee::salary = salary;
  else // Nedopustimihy oklad
```

```
employee::salary = 0.0;
```

Destruktorlar

Sinfning biror ob'ekti uchun ajratilgan hotira ob'ekt yo'qotilgandan so'ng bo'shatilishi lozimdir. Sinflarning mahsus komponentalari destruktorlar, bu vazifani avtomatik bajarish imkonini yaratadi.

Destruktorni standart shakli qo'yidagicha :

~ sinf_nomi ( ) {destruktor tanasi}

Destruktor parametri yoki qaytariluvchi qiymatga ega bo'lishi mumkin emas. (hatto void tipidagi)

70 - DARS. STATIK ELEMENTLAR. SINF STATIK KOMPANENTALARI .

Sinf kompleks ob'ektlari uchun umumiy bo'lgan elementlar statik elementlar deb ataladi. Yangi ob'ektlar yaratilganda statik elementlarga murojat qilish uchun oldin inizializatsiya qilinishi lozim. Inizializatsiya qo'yidagicha amalga oshiriladi:

Sinf-nomi:: kompleks-nomi initsializator

Misol uchun skladdagi tovarni kompleks tasvirlovchi sinfni kurib chiqamiz. Bu sinf komponentalari qo'yidagilardan iborat:

- Tovar nomi
- Olish narhi
- Kushimcha narh foiz ko'rinishida
- Tovar haqida ma'lumotlar kiritish funktsiyasi
- Tovar haqida ma'lumotlar va Tovar narhini chiqaruvchi funktsiya;

Sinf ta'rifi :

Goods. Cpp

```
#include <iostream.h>
Struct goods
{ char name [40];
float price;
Static int percent;
Void input()
{cout <<" Tovar nomi:"; cin>>name;
  cout<<" Tovar narhi:";cin>>price;
}
```

Har bir yangi ob'ektning kompanentalari faqat shu ob'ektga tegishli bo'ladi . Sinf kompanentasi yagona bo'lib va hamma yaratilgan ob'ektlar uchun umumiy bo'lishi uchun uni statik element sifatida ta'riflash ya'ni Static atributi orqali ta'riflash lozimdir . S sharning statik kompanentalarini inizializatsiya qilishdan so'ng dasturda ob'ektlarni kiritmasdan oldin ishlatish mumkin. Agar kompanenta private yoki protected sifatida ta'riflangan bo'lsa unga kompanenti funktsiya yordamida murojat qilish mumkin. Lekin kompaneta funktsiyaning chaqirish uchun biror ob'ekt nomini ko'rsatish lozim. Lekin statik kompanentaga murojat qilinayotgan birorta ob'ekt yaratilmagan bo'lishi yoki bir nechta ob'ektlar yaratilgan bo'lishi mumkin shuning uchun sinf statik elemntlariga ob'ekt nomini ko'rsatmasdan murojat qilish imkoniyatiga ega. Bunday imkoniyatni statik kompanenta funktsiyalar yaratadi. Statik kompanenta funktsiyaga ob'ekt nomi yoki obe'ktga ko'rsatkich orqali murojaat qilish mumkin . Shu bilan birga nomi orqali qo'ydagicha murojaat qilish mumkin.

Sinf - nomi : Statik – funktsiya –nomi

Qo'yidagi dasturda point sinfi uch o'lchovli fazodagi nuqtani aniqlaydi va shu bilan birga o'z ichiga shu nuqtalar sonini oladi.

```
# include <iostream. h >
clearr point 3
{double x,y,z;
static int N;
public
point 3 (double xu=0.o,double yu=0.o, double zu=0.o)
{N + +; x=xn; y=yn; z=zn; }
static int & count ( ) {return N; }
};
```

```
int point 3: :N=0;
void main (void)
{cout < <"\ n size of ( point 3)=" < < size of (point 3) ;
point 3 A (0: 0, 1. 0, 2. 0);
cout < < " \ nsize of (A)="< < size of (A)
point 3 B (3.0, 4.0, 5.0)
```

Ma'lumot elementlardan birgalikda foydalanish.

Agar bir sinf ob'ektlari umumiy ma'lumotlardan(statik elementlardan foydalanish zarur bo'lishi mumkin. Bu hollarda elementlarni static hizmatchi so'zi yordamida ta'riflash lozimdir:

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

Sinfni ta'riflagandan so'ng elementni sinf tashqarisida global o'zgaruvchi sifatida ta'riflashingiz lozim:

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

Kuyidagi SHARE_IT.CPP dasturida book_series, sinfi aniqlangan bo'lib, bu sinfga tegishli ob'ektlar uchun bir hil bo'lgan page_count, elementidan foydalanilgandir. Agar dasturda bu element qiymati o'zgartirilsa hamma ob'ektlarda ko'rinadi:

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

```
}
void main(void)
{
   book_series programming( "Uchimsya programmirovat' na C++", "Jamsa", 22.95);
   book_series word( "Uchimsya rabotat' s Word dlya Windows", "Wyatt", 19.95);
   word.set_pages(256);
   programming.show_book ();
   word.show_book() ;
   cout << endl << "Izmenenie page_count " << endl;
   programming.set_pages(512);
   programming.show_book();
   word.show_book();
}
```

Bu dasturda sinf page_count elementini static int. shaklda ta'riflaydi. Sinf ta'rifidan so'ng page_count elementi global uzgaruvchi sifatida ta'riflanadi. Dastur page_count, elementi qiymatini o'zgartirganda, bu o'zgarish book_series sinfining hamma ob'ektlarida ko'rinadi.

Ob'ektlar mavjud bo'lmaganda public static tipidagi elementlardan foydalanish Statik elementlardan ob'ekt hali yaratilmasdan foydalanish mumkindir. Buning uchun bu elementni public va static sifatida ta'riflash lozimdir. Quyidagi misolda USE_MBR.CPP dasturi book_series sinfiga tegishli page_count elementini thu ob'ektga tegishli ob'ektlar mavjud bo'lmasa ham ishlatadi:

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

Bu misolda page_count elementi public sifatida ta'riflangani uchun, book_series sinfi ob'ektlari mavjud bo'lmagan holda ham dastur bu elemenetga murojaat qilishi mumkin. Statik funktsiya elementlardan foydalanish Dasturda statik elementlardan tashqari statik funktsiyaldaridan ham foydalanish mumkindir. Bu holda shu sinfga tegishli ob'ekt yaratilmasa ham dastur statik funktsiyaga murojaat qilishi mumkindir. Quyida menu sinfi ta'riflangan bo'lib ANSI drayvera esc- ketma ketligi yordamida ekranni tozalash uchun ishlatadi. Buts sinfning clear_screen usuli statik usul sifatida ta'riflangani uchun menu tipidagi ob'ektlar mavjud bo'lmasa ham bu usuldan foydalanish mumkindir. Quyidagi
CLR_SCR.CPP dasturda clear_screen usulidan foydalanish ko'rsatilgan:

```
#include <iostream.h>
class menu
{
public:
   static void clear_screen(void);
   // Zdes' doljnih biht' drugie metodih
private:
   int number_of_menu_options;
};
void menu::clear_screen(void)
{
   cout << '\033' << "[2J";
}
void main(void)
{
   menu::clear_screen();
}
```

Dasturda clear_screen elementi statik , element sifatida ta'riflangani uchun, menu tipidagi ob'ektlar mavjud bo’lmasa ham bu funktsiyadan ekranni tozalash uchun foydalanish mumkindir.

71 - DARS. SINFLAR VA KO'RSATKICHLAR.

Sinf kompanentalariga ko'rsatkichlar.

Sinfga tegishli funktsiyaga ko'rsatkichlar qo'yidagi ko'rinishga ega:

Funktsiya-tipi (sinf-nomi : :* kursatkich-nomi)
(formal parametrlar spetsifikatsiyasi)

Masalan complex sinfida double & ze ( ), double & im ( ) metodlari aniqlangan. Sinf tashqarisida Ptcom ko'rsatkichini qo'yidagicha kiritish mumkin.

```
Double & (complex : :* Pt Com) ( );
```

Bu ko'rsatkich qiymatini oddiy usulda berish mumkin:

```
Pt Com=& complex : : v;
```

Endi complex sinfiga tegishli A ob'ekt uchun:

```
Complex A (10.0, 2.4);
```

Shu sinfga tegishli re( ) funktsiyasini quyidagicha chiqarish mumkin.

```
(A.     * PtCom) ( )=11.1;
cont < < (A. * PtCom ) ( );
```

Ko'rsatkich qiymatini o'zgartiramiz:

```
PtCom=& complex : : im;
```

Endi bu ko'rsatkich yordamida shu sinfning boshqa funktsiyasining chaqirish mumkin:

```
Cont < < (A. ^ PtCom) ( );
Complek B=A;
(B.     ^ PtCom ( ) f=3.0;
```

Sinfining komponenta ma'lumotlariga murojat qiluvchi ko'rsatkich ta'rifi:

Ma'lumot-turi (sinf-nomi : : * kursatkich-nomi);

Ko'rsatkichni ta'riflashda initsializatsiya qilish mumkin, lekin buning uchun kompanenta public (umumiy) formatda ega bo'lishi kerak. Masalan: satr uzilishiga ko'rsatkich yaratish:

```
Int (stroka : : * pllu)=&  stroka : : len;
```

Tenglikka olib keladi, chunki len kompanentasi private atributiga egadir. Sinf kompanentasiga ko'rsatkichni funktsiyalarni chaqirishda haqiqiy parametr shartida ishlatish mumkin. Buning uchun bu parametrlari qo'yidagiga murojaat qilish lozim.

Ob'ekt-nomi.* kompanenta-ma'lumotga-ko'rsatkich
Ob'ekt-nomi.* metodga-ko'rsatkich (parametrlar)

Misol uchun complek sinfi ob'ektlariga ko'rsatkich kiritamiz;

```
Complex 1  CM(10.2,-6.4);
Complex 1  * PCOM1=& CM;
```

Bu holda

```
PCOM1-* PCM1=22.2
```

Keltirilgan operator M ob'ekt sifatida kiritilgan kompleks sonini haqiqiy qismini o'zgartiradi. Qo'yidagi misolda mos metodga murojaat bajariladi:

```
Complex A (22.2, 33.3);
complex * P Complex=&A;
Void (complex : : * dirplay) ( );
Pdi 1 play-& complex : : di 1 play;
(P Complex -* Pdi 1 play ( );
```

Bu misolda ekranga qo'yidagi ma'lumot chiqariladi:

```
Real=22.2; imag=33.3;
Void di 1 play (void)
{cont < <"\ n satr uzunligi: " < < ch;
```

```
}
~ stroka ( ) { delete [ ] ch; }
};
# include “stroca C P P”
void main ( )
{ stroka LAT (“Non Multa, sed Multa”);
   stroka RUS (“Ne mnogo; no mnogoe”);
   stroka CTP (20);
   LAT. Display( );
   Count < < “\ n B ob'ekt RUS: “ << Run string ( );
    C T P. display ( );
}
```

Stroka sinfi clarr hizmatchi so’zi yordamida kiritilgani uchun char * ch va int len kompanentalariga to’g’ridan to’g’ri murojaat qilib bo’lmaydi. Satr uzunligini aniqlash uchun len-str ( ) kompanenta funktsiyasiga murojaat qilish lozimdir. Konkret ob'ektga tegishli satrga ko’rsatkichni string ( ) funktsiyasi qaytaradi. Sinfning kerakli konstruktori mavjud bo’lib qo’shimcha yuklangan funktsiyalardir. Har bir konstruktor bajarilganda dinamik hotira ajratiladi. Sinf destruktori ~stroka ( ) bu hotirani ozod qiladi. in+N parametrlari konstruktor chaqirilganda N+1 elementli massiv bo’sh qoladi,
Satr uzunligi bo’lsa 0 ga teng bo’ladi.
Char * orch parametrli konstruktor chaqirilganda, massiv uzunligi va ---.

72 - DARS. THIS KO'RSATKICHI.

Sinfga tegishli funktsiya aniq ob'ekt ma'lumotlarini qayta ishlash uchun chaqirilganda bu funktsiya avtomatik ravishda ob'ektga ko'rsatkich uzatiladi. Bu ko'rsatkich belgilangan this nomiga ega va dasturchi uchun bilintirmagan holda sinf har bir funktsiyasi uchun quyidagicha aniqlangan:
Sinf-nomi const this=ob'ekt adresi; this hizmatchi so'zini ta'riflash lozim emas. Sinf komponentalariga sinfga tegishli funktsiyalarda murojat qilintanda shu ko'rsatkichdan foydalanish mumkin.
Misol uchun sinfni qo'yidagicha ta'riflash mumkin:

```
Stact s,s
{ int si; char sc;
ss(int in, char cn )
{this ->si =in; this ->sc=cn;}
void print(void)
{ cont<<"\n si="<< this ->si;
cont<<"\n sc="<< this ->sc;
```

Bu misolda this ko'rsatkichidan foydalanish hech qanday afzallikka ega emas. Qo'yidagi misolda sinf komponentasi nomi va sinfga tegishli funktsiya formal parametri nomi bilan ustma ust tushadi.
Bu holda this ko'rsatkichidan foydalanish lozimdir.

```
#include <iostem.h>
class cell
{ int static Amount;
int Namber;
double Meaning;
public:
cell(double Meaning=0.0)
{Amount tt;
this ->number=Amount ;
this-> Meaning=Meaning;
}
void display (void)
{ cont<<"\Number="<<this->Number;
cont<<"\Amount="<<this->Amount;
cont<<"\Meaning="<<this->Meaning;
}
};
int cell::Amount=0
void main(void)
{ cell A;
A.display();
Cell B(200.0);
Cell C(300.0);
B.display();
C.display();
}
```

73 - DARS. SINFLAR DO'STLARI.

Sinfning kompanentalariga murojat qilishning yana bir usuli do'stona funktsiyalardan foydalanishdir. Sinfning do'stona funktsiyasi deb shu sinfga tegishli bo'lmagan lekin shu sinfning himoyalangan kompanentlariga murojat qilish huquqiga ega bo'lgan funktsiyalarga aytiladi. Funktsiya do'stona bo'lishi uchun sinf tanasida friend spetsifikatori bilan ta'riflanishi lozim.
Do'stona funktsiyaga ega bo'lgan sinfga misol:

```
# include <conio.h>
cla 11 charl 0 cu 1
int x,y;
char c c;
friend void friend_put (char locu1*, char);
public:
char locu1 (int xi, int yi, char ci)
{ x=xi; y=yi; cc=ci; }
void dirplay (void)
{ gotoxy (x,y); putch (cc); }
void friend_put (char locu1 * p, char c)
{ p-cc=c; }
void main (void)
{ char locu1 D (20,4, `d`);
char locu1 S (10,10, `s`);
clrrcr ( );
D. di 1 play ( ); getcr ( ); S. di1play ( ); getch( );
Friend_put (& D, `x`); D. dirplay ( ); getch( );
Friend_put (& S, `#`); S. di 1 play( ); getch( ); }
```

Dasturda D va S ob'ektlari yaratilib ular uchun ekranda koordinatalar va (d,s) simvollari aniqlanadi. Shundan sung sinf funktsiyasi char locu1 : : di 1 play ( ) simvollarni ko'rsatilgan pozitsiyaga chiqaradi. Global friend_put funktsiyasi simvollarning urnini almashtirib qo'yadi. Do'stona funktsiyalardan foydalanish hususiyatlari qo'yidagilardir.
Do'stona funktsiya murojaat qilinganda this ko'rsatkichiga ega bo'lmaydi. Sinf ob'ektlari do'stona funktsiyaga parametrlari orqali uzatilishi lozim. Do'stona funktsiya sinf kompanentasi bo'lmagani uchun unga tanlov amalini qo'llab bo'lmaydi:

Sinf ob'ekti .Funktsiya nomi va ob'ektga_kursatkich-funktsiya nomi.

Do'stona funkiyaga murojaat spetsifikatorlari (public, protected, private) qo'llanmaydi. Do'stona funktsiya prototipining sinf usulida joylashtirilishi farqi yo'q. Do'stona funktsiyalar mehanizmi sinflar orasidagi aloqani soddalashtirishga imkon beradi. Sinflardan berkitilgan kompanentalariga murojaat qilish uchungina kiritilgan funktsiyalarni olib tashlash mumkin.
Misol tariqasida "sohadagi nuqta" va "sohadagi chiziq" sinflari uchun do'stona funktsiyani qarab chiqamiz. Sohadi nuqta sinfiga, (h,u) koordinatalarini aniqlovchi kompanentalar kiradi. Sohadagi chiziq sinfining kompanentalari chiziqning umumiy tenglamasi A*h+V*u+S=0 tenglamasi koeffitsentlari A,V,S. Qo'yidagi dasturda ikkala sinf uchun do'stona bo'lgan nuqtadan chiziqqacha masofani hisoblashga imkon beradigan funktsiya kiritilgan.

```
# include <io 1t ream.h>
cla11 line 2 ;
```

```
cla11 point 2
{ float x,y ;
public :
point 2 (float xn=0, float yn=0)
{ x=xn; y=yn; }
friend float uclou (point, line 2);
}
cla11 line 2
float A,B,C;
public:
line 2 (float a, float b,float c)
{A=a; B=b; C=c;}
friend float uclou (point 2, line 2);
}
float uclou (point 2 p, line 2 1)
{return 1.A-P.x+1.B*P.y+1.c;)
void main (void)
{ point 2 P(16.0,0.12,12.3);
line 2 h (10.0,-42.3,24.0);
cout << "\n Uklonenie tochik R ot pryamoy L: ";
cout << uclon (P,L);
}
```

Dastur bajarilishi natijasi

R nuqtadan L chiziqqacha masofa: -336.29009

Bir sinf ikkinchi sinfga do'stona bo'lishi mumkin. Bu holda sinfning hamma komponenta funktsiyalari boshqa sinfga do'stona bo'ladi. Do'stona sinf o'zga sinf tanasidan tashqari ta'riflangan bo'lishi lozim. Masalan:

```
Cla11 X 2 { friend clarr X1;... };
Cla11 X 1 {...
        Void f1 (...);
        Void f2 (...);
        ...
        };
```

Bu misolda f1 va f2 funktsiyalar X2 sinfiga do'stonadir. Sinflar orasidagi do'stlikka misol tariqasida "N-o'lchovli fazodagi nuqta"-Point N va unga do'stona bo'lgan "Nuqta radiusi vektori"-olctorN sinflarini ko'rib chiqamiz. Sinf hamma kompanentalari-o'lchovi N point va koordinatalar massivi X[Point] hususiydir, va shuning uchun ularga do'stona funktsiya orqali murojat qilish mumkin.

Point N sinfi konstruktori koordinatalari massivi berilgan parametrlar qiymatlari bilan initsializatsiya qiladi. Vector N sinf konstruktor "vektor"ob'ektini Point N sinfining ikki ob'ekti buyicha aniqlaydi. Point N sinfi ob'ektlari vektorning boshi va ohirini aniqlaydilar. Bu vektor koordinatalar boshiga keltiriladi. Konstruktordan tashqari vector N sinfiga vektor normasini hisoblovchi funktsiya kiritilgan. Vektor normasi nuqtalarining koordinatalari yigindisidir. Dasturda ikki ulchovli ikki nuqtasi bo'yicha vektor hosil qilingan va so'ngra har hil o'lchovli ikki nuqtadan vektor hosil qilishga harakat qilingan.

```
# include < io 1t ream.h >
# include < stdlib.h >
cla11 Point N
int N point;
double*x;
```

```
friend cla11 vectorn N;
public:
Point N (int n, donble d=0. 0)
{ N point : : point N (int n, double d)
{ N point=n;
x=new double [ N point ];
for (int I=0; I<N point; itt)
x [I]=d;
}
cla11 vector N
double* x0;
int N vector;
public;
vector N (point N, point N);
double norm, ( );
}
vector N: : vector N (pointN beg, point.N end)
{ ij (beg. N point!=end. N point)
{ cerr << “\n nukta ulchovida hato “)
exit (1)
}
N vector=beg. N point;
Xv=nev double [N vector];
For ( int I=0; I<N vector; iff )
Xv [I]=end. x[I]-beg.DC [I]
}
double vector N: : norm ( )
{double dd=0.0
for (int I=0; I<N vector; iff)
llf=xv[I]*xv[I];
return dd;
}
void main (void)
{ point N A (2,4,0);
point N B (2,2.0);
vectorn V (a,b)
count < c “\n Vektor normasi : “<< v. Norm ( );
point N X(3,2.0)
vectorn J (A,X);
}
```

Dastur natijasi:
Vektor normasi 8
Nuqta o’lchovida hato!

Sinf do’stlari

C++ biror sinf do’stlariga shu sinfning hususiy elementlariga murojaat qilishgsha imkon beradi.Misol uchun qo’yida keltirilgan book sinfi librarian sinfini o’zining do’sti deb e'lon qiladi. Shuning uchun librarian sinfi ob'ektlari to’g’ridan to’g’ri book, sinfining hususiy elementlariga murojaat qilishlari mumkin ispol'zuya:

```
class book
```

```
{
public:
   book (char *, char *, char *);
   void show_book(void);
   friend librarian;
private:
   char title [64] ;
   char author[64];
   char catalog[64];
};
```

Qo'yidagi VIEWBOOK.CPP dasturida librarian sinfi book sinfini uz do'sti deb e'lon qiladi.Dasturda librarian sinfining change_catalog funktsiyasidan foydalanilgan:

```
#include <iostream.h>
#include <string.h>
class book
{
public:
   book (char *, char *, char *);
   void show_book(void);
   friend librarian;
private:
   char title[64] ;
   char author[64];
   char catalog[64];
};
book::book(char *title, char *author, char •catalog)
{
   strcpy(book::title, title);
   strcpy(book::author, author) ;
   strcpy(book::catalog, catalog);
}
void book::show_book(void)
{
   cout << "Nazvanie: " << title << endl;
   cout << "Avtor: " << author << endl;
   cout << "Katalog: " << catalog << endl;
}
class librarian
{
public:
   void change_catalog(book *, char *);
   char *get_catalog(book);
};
void librarian::change_catalog(book *this_book, char *new_catalog)
{
   strcpy(this_book->catalog, new_catalog);
}
char *librarian: :get__catalog(book this_book)
{
   static char catalog[64];
```

```
  strcpy(catalog, this_book.catalog);
  return(catalog) ;
}
void main(void)
{
  book programming( "Uchimsya programmirovat' na yazihke C++", "Jamsa", "P101");
  librarian library;
  programming.show_book();
  library.change_catalog(&programming, "Legkiy C++ 101");
  programming.show_book();
}
```

Dustlar sonini chegaralash

C++ tilida do'stona sinfning hamma funktsiyalari balki ba'zi funktsiyalariga hususiy elementlarga murojaat huquqini berish mumkindir. Misol uchun librarian sinfining change_catalog va get_catalog funktsiyalariga book sinfing hususiy elementlariga murojaat qilish imkonini berish lozim bo'lsin. U holda book sinfida bu huquqni qo'yidagicha ko'rsatish mumkindir:

```
class book
{
public:
  book(char *, char *, char *);
  void show_book(void);
  friend char *librarian::get_catalog(book);
  friend void librarian: :change_catalog( book *, char *);
private:
  char title[64];
  char author[ 64 ];
  char catalog[64];
};
```

Bu misoldan ko'rinib turibdiki friend operatorlari do'stona funktsiyalarning Tula prototiplariga egadirlar.Dustona funktsiyalar. Agar dasturda bir sinfdan ikkinchi sinfga murojaat mavjud bo'lsa va sinf ta'riflari tartibi notug'ri keltirilgan bo'lsa sintaksis hato kelib chiqadi. Misolda book sinfi librarian sinfida aniqlangan funktsiyalar prototiplaridan foydalanadi. Shuning uchun librarian sinfi ta'rifi book sinfi ta'rifidan oldin kelishi kerak. Lekin librarian sinfi tahlil qilinsa u book sinfiga murojaat qilishini aniqlash mumkin:

```
class librarian
{
public:
  void change_catalog(book *, char *);
  char *get_catalog(book);
};
```

Dasturda book sinfi ta'rifini librarian sinfi ta'rifidan oldin qo'yish mumkin bo'lmaganligi uchun, C++ oldin book sinfini e'lon qilib so'ngra keyinroq ta'riflash imkonini beradi. Qo'yida qanday amalga oshirish mumkinligi ko'rsatilgan:

```
class book; //
```

Qo'yidagi LIMITFRI.CPP dasturda do'stona funktsiyalarodan foydalanilgan. Sinflar tap'riflari tartibiga e'tibor bering:

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

```
#include <string.h>
class book;
class librarian
{
public:
  void change_catalog(book *, char *);
  char *get_catalog(book);
};
class book
{
public:
  book(char *, char *, char *) ;
  void show_book (void);
  friend char *librarian::get_catalog(book);
  friend void librarian::change_catalog( book *, char *);
private:
  char title[64];
  char author[64];
  char catalog[64];
};
book::book(char *title, char *author, char *catalog)
{
  strcpy(book::title, title);
  strcpy(book::author, author);
  strcpy(book::catalog, catalog);
}
void book::show_book(void)
{
  cout << "Nazvanie: " << title << endl;
  cout << "Avtor: " << author << endl;
  cout << "Katalog: " << catalog << endl;
}
void librarian::change_catalog(book *this_book, char *new_catalog)
{
  strcpy(this_book->catalog, new_catalog) ;
}
char *librarian::get_catalog(book this_book)
{
  static char catalog[64];
  strcpy(catalog, this_book.catalog);
  return(catalog) ;
}
void main(void)
{
  book programming( "Uchimsya programmirovat' na C++", "Jamsa", "P101");
  librarian library;
  programming.show_book();
  library.change_catalog(&programming, "Legkiy C++ 101");
  programming.show_book();
}
```

Dasturda book sinfi mavjudligi e'lon qilinadi. Shuning uchun librarian sinfi hali ta'riflanmagan sinfga murojaat qilishi mumkin.

74 - DARS. SINFLAR DO'STLARI.AMALLARNI KAYTA YUKLASH STANDART AMALLARINI QAYTA YUKLASH.

S ++ tilining ajoyib hususiyatlaridan biri standart amallarni yangi ma'lumotlar turlariga kullash imkoniyatidir. Masalan satrlani ulashni S1=S2 kurinishda belgilash ancha ulaydir. Bu amalni simvolli massivlarga yoki satrli konstantalarga qo'llashning iloji yo'q. Lekin S1 va S2 ni biror sinf ob'ektlari sifatida (masalan Stroka sinf) tavsiflansab ular uchun amalini kiritish mumkin bo'ladi. Amalni ma'lumotlarning yangi tipiga qo'llash uchun dasturchi " operatsiya – funktsiya " deb ataluvchi mahsus funktsiyani kiritishi lozim. Operatsiya – funktsiya ta'rifi qo'yidagicha.

Qaytariluvchi_ma'lumot_tini operator operatsiya_belgisi
(operatsiya-funktsiya-parametrlari-spetsifikatsiyasi)
{ operatsiya-funktsiya-tanasi-operatorlari }

Kerak bo'lganda bu funktsiya operator prototipini kiritish mumkin.

Qaytariluvchi-ma'lumot-tipi+operator operatsiya-belgisi
{ operatsiya-funktsiya-parametrlari-spetsifikatsiyasi)

Misol uchun * amalni biror T sinfga tegishli ob'ektlarga qo'llash uchun qo'yidagicha funktsiya e'lon qilishi mumkin:

```
T operator * ( Tx,Ty )
```

Bu usulda ta'riflangan operatsiya qo'shimcha yuklangan (inglizchasiga-overload) deb ataladi. Agar T sinf uchun yuqorida keltirilgan turdagi funktsiya e'lon qilingan bo'lsa, A*V ifoda operator (A,V) sifatida qaraladi. Sinf ob'ektlariga funktsiya-operatorni qo'llash uchun operatsiya-funktsiya yoki sinf kompanenta funktsiyasi yoki do'stona funktsiya bo'lishi, eki parametrlardan birortasi sinf tipiga ega bo'lishi kerak. Qo'yidagi misolda amalini satrlarga qo'llash usuli kursatilgan. Buning uchun len – satr uzunligi , ch-simvolli massivga ko'rsatkich parametrlardan iborat. Streko sinfidan foydalanamiz. Bu sinfda ikki konstruktor mavjuddir. Birinchisi yaratilaetgan ob'ekt uchun hotiradan satr sifatida joy ajratadi. Ikkinchisi haqiqiy parametr asosida ob'ekt yaratadi. Sinfdan tashkarida operatsiya-funktsiyani qo'yidagicha S bilan aniqlaymiz:

Strka & operator + (stroka & A, stroka & B),

Bu operatsiya-funktsiya '+' amalini stroka sinfiga tegishli ob'ektlarga qo'llash uchun ishlatiladi. Funktsiya operatsiya tanasi asosiy dasturdan keyin keltirilgan, shuning uchun asosiy dasturda bu funktsiyaning prototipi joylashtirilgan.

```
# include " stroka. Spp "
stroka & operator + (stroka & A, stroka & B);
void main (void)
{ stroka X ("qui");
stroka Y ("vivra");
stroka J ("verra");
stroka C;
c=x+Y+J+"-Tirik bulsak, ko'ramiz".;
c. di 1 play ( );
}
stroka 2 operator + (stroka &, stroka & b)
{ int ii=a. len-str ( ) + b.len-str ( );
stroka * ps;
ps=new stroka (ii);
strcpy (ps-string ( ), a.string ( ) );
strcut (ps-string ( ),b.string ( ) );
```

```
ps-len_str ( )=ii;
return * ps;
}
```
Dastur bajarilishi natijasi:
Satr uzunligi 36
Satr mazmuni: Qui Vivra Verra –Tirik bo'lsak quramiz!
Dasturda satrlar uchun kiritilgan '+' amali bitta ifodada uch marta qo'llanadi:

X+Y+J f"-Tirik bo'lsak qo'ramiz!"

Bunday qisqa murojatdan tashqari to'la operatsiya funktsiyani chiqarish mumkin:

C= operator t (X,Y);
C= operator t (C,J);
C= operator t (C," Tirik bo'lsak, quramiz!")

Ikkinchi imkoniyat sinf kompanenta funktsiyalardan foydalanishdir. Har qanday biror amal sinfga tegishli statik kompanenta operatsiya-funktsiya yordamida qayta yuklanishi mumkin. Bu holda bitta parametrga ega bo'lib, sarlavhasi qo'yidagi ko'rinishda bo'ladi:

T operator & (T.X)

Bu erda T-sinf, &-operatsiya.

Bu holda A&V ifoda A. Operator & (B) murojaat sifatida talqin yotiladi. Kerak bo'lganda this ko'rsatkichi orqali murojat qilish mumkin. Misol tariqasida 't' amalini point sinfi bilan ta'riflanuvchi displeydagi nuqtalarga qullaymiz. Soddalashtirish uchun point sinfida eng kerakli kompanentalarni qoldiramiz.

```
        # include < graphic1. h >
        cla11 point1
        protected:
        int x,y;
        public:
        point1 (int xi=0, int yi=0)
        { x=xi; y=yi; }
        void show (void) { putpinel (x,y,get color ( ) );};
        point1 operator+ (point2 p);
        };
        point1 point1 : : operator + (point &p)
        { point1: d;
        d.x=thi1-x+p.x;
        d.y=thi1-y+p.y;
Return d;
        }
        # include <conio.h>
        ooid main ( )
        {
        int dr=DETECT, mod;
        point1 A(200,50);
        point1 B;
        point1 D(50,120);
        INITYRAPH (& DR, & MOD, "C\\borlandc \\ BG$");
A.      show ( );
getch ( );
B.      show ( ); getch ( );
D show ( ); getch ( );
B=A+P;
```

```
B.      show ( ); getch ( );
B=A. operator t (B);
B.      show ( ); getch ( );
closegraph ( );
}
```
Dastur bajarilishi natijasida display ekraniga ketma-ket qo'yidagi nuqtalar qo'yiladi: A(200,50); B(0,0); D(50,120); B(250,70), B(450,220)

Operatsiya funktsiyani oddiy sinf komponenta funktsiyasi sifatida chaqirish mumkin:

```
Point1 * ptr=& A;
B=ptr – operator + (D);
```

Biz sinf amalini global operatsiya-funktsiya va kompanenta operatsiya-funktsiya yordamida qayta yuklashni ko'rib chiqdik. Endi unar amalni sinf do'stona funktsiyasi yordamida qayta yuklashni ko'rib chiqamiz. "N ulchovli fazo radius-vektori " sinfini kiritamiz va bu sinf uchun '-'-unar operatsiya funktsiyani kiritamiz. Bu operatsiya vektor yunalishini teskarisiga o'zgartiradi.

```
# include < iostream. h >
class vector
int N;
double * x;
friend vector 2 operator – (vector &);
public:
vector (int n, double * xn)
{ N=n; x=xn; }
void di 1 play ( );
};
void vector : : di 1 play ( )
{ cont < < "\n Vektor koordinatalari :";
for ( int I=0; I<N; iff )
cont < < "\t"  < < x [I];
}
vector 2 operator –(vector & 0)
{ for ( int I=0; I<V.N; iff )
V. x[I]=-V. x[I];
Retun v;
}
```

Qo'yidagi kompanentalarni qayta yuklash mumkin emas.

strukturalangan ob'ekt kompanentasini to'g'ridan to'g'ri tanlash.

#komponentaga ko'rsatkich orqali murojaat qilish;

:shartli operatsiya:

:: ko'rinish doirasini aniqlash;

Sizeof-hotira hajmini aniqlash ;

preprotsessor direktivasi;

protsessorli amal;

Qayta yuklash mehanizmi yana quyidagi hususiyatlarga ega:

Standart amallarni qo'shimcha yuklanganda prioritetlarini o'zgartirish mumkin emas.

Qo'shimcha yuklangan amallar uchun ifodalar sintaksisini o'zgartirish mumkin emas. Unar yoki binar amallarni kiritish mumkin emas.

Amallar uchun simvollar kiritish mumkin emas masalan kupaytirish uchun **belgisi.

Har qanday binar amal ikki usul bilan aniqlanadi, yoki bir parametrli kompanenta funktsiya sifatida yoki global yoki do'stona global ikki parametrli funktsiya. Birinchi holda x*y ifoda h. Operator*(y) murojaatni ikkinchi holda esa Operator*(x*y) murojaatni bildiradi.

Binar '=', '[ ]', '->' amallar semantikasiga ko'ra Operator=, Operator[ ], Operator-> global funktsiya bo'lolmaydi. Balkim nostatik kompanenta funktsiyasi bo'lishi lozim.

Har qanday amal '$' sinf ob'ektlari uchun ikki usulda aniqlanadi yoki parametrsiz komponenta funktsiya yoki bir parametrli (balkim do'stona) global funktsiya. Prefiks amal uchun xz ifoda , postfiks amal uchun Z ifoda komponenta funktsiya z.operator*( ) yoki global funktsiya operator*(z) chaqirilishini bildiradi. C++ tilida ba'zi amallarni boshqa amallarning kombinatsiyasi sifatida aniqlanadi.

Misol uchun j+m butun son uchun m+=1ni bu amal bo'lsa m=m+1 ni bildiradi. Bunday avtomatik almashtirishlar qo'shimcha yuklangan amallar uchun bajarilmaydi. Misol uchun umumiy holda operator*=() ta'rifni operator*( ) ta'rif va operator=( ) ta'rifdan keltirib chiqarib bo'lmaydi.

Agar ifodada foydalanuvchi kiritgan sinf ob'ekti qatnashmasa uning ma'nosini o'zgartirib bo'lmaydi. Misol uchun faqat ko'rsatkichlarga ta'sir qiluvchi amallarni kiritish mumkin emas.

Agar operatsiya funktsiyaning birinchi parametri standart tip bo'lishi kerak bo'lsa, bunday operatsiya-funktsiya kompanenta-funktsiya bulolmaydi. Misol uchun aa- biror sinf ob'ekti bo'lsin va uning uchun '+' amali qo'llangan bo'lsin. AA+2 ifoda uchun yoki AA.operator(2) yoki operator+(AA,2) ifoda chaqirilishi mumkin. 2++AA ifoda uchun operator+(AA,2) chaqirilishi mumkin lekin z. operator+(AA) hatoga olib keladi. Amallar kengaytirilganda ular uchun har hil tiplar qolib inatsiyasini oldindan nazarda tutish lozim. Lekin operatsiya-funktsiyalarga murojaat qilinganda standart tiplar almashinuvchi qoidalari ishlatiladi, shuning uchun tiplarning hamma kombinatsiyalarini hisobga olish zarurati yuq. Kupgina hollarda binar amallar uchun qo'yidagi hollarni hisobga olish etarlidir.

```
# standart tip, cinf
# sinf, standart tip
# sinf, sinf
```

Masalan: Somplex sinfi uchun qo'yidagi do'stona operatsiya-funktsiyalarni kiritish mumkin:

```
Complex operator++ (Somplex, Somplex y)
{Return (Somplex(x.real+y.real, x.imag+y.imag());
Complex operator+(Complex x, double y}
{Return (Complex (x.real+y, x.imag ())}
```

Shundan sung qo'yidagi ifodalarni qo'llash mumkin bo'ladi:

```
Complex CC (1.0, 2.0);
Complex EE;
EE=4.0+CC;
EE=EE+2.0;
EE=CC+EE;
CC=EE+'e';
```

Standart tiplarni sinf ob'ektiga keltirish vazifasini konstruktorga topshirish mumkin. Masalan Complex sinfiga qo'yidagi konstruktorni qo'yish hamma yordamchi funktsiyalardan halos bo'lish imkonini beradi:

```
Complex (double x);
{Real=x; imag=0.0;);
```

Bu holda qo'yidagi prototipga ega bo'lgan do'stona operatsiya funktsiyadan foydalanish etarli.

```
Friend Complex operator+ (Complex, Complex);
```

Sinfga konstruktor qo'shish o'rniga yagona konstruktrga ikkinchi parametr qiymatini kiritish etali:

```
Complex (double re, double im=0.0)
{Real=re; imag=im;}
```

++ va amallari prefiks va postfiks shakllariga ega bo'lgani uchun bu amallarni qo'shimcha yuklash o'ziga hos hususiyatlarga ega.Misol uchun qo'yidagi programmada ++ amali pair sinfiga tegishli ob'ektlarga bir parametrli do'stona operatsiya-funktsiya yordamida qo'shimcha yuklangan:

```
Friend pair & operator ++ (pair&):
```

--amali Pair sinfining parametrsiz kompanenta funktsiyasi yordamida qo'shimcha yuklangan:

```
Pair&pair: operator--();
```

C++ kompilyatorida. C++ tilining eng birinchi varianti amalga oshirilgan shuning uchun bu kompilyator prefiks va postfiks shakllarni ajratmaydi:

```
Class pair
{Int N
Double x;
Friend pair& operator++(pair);
Public:
Pair (int n, double xn)
{N=n; x=xn;}
Vouid display ()
{Cont <c"/n Kordinatalar: N="<<N<<"/+x="<<x;
}
Pair& operator-- ()
{N-=1; x-=1.0;
Return this;}
Pair& operator++(pair& p)
{P.N+=1; P.x+=1.0;
Return P;
}
Void main ()
{Pair z (10,20.0);
Z.display ();
++Z;
Z.display ();
--Z;
Z.display ();
Z++;
Z.display ();
Z--;
Z.display ();
}
```

TC++da programma bajarilishi natijalari:

```
Kordinatar: N=10 x=20
Kordinatar: n=11 x=21
Kordinatar: n=10 x=20
```

Kordinatar: N=11 x=21
Kordinatar:N=10 x=20

C ++ tilining zamonaviy versiyalarida prefiks ++ va -- operatsiyalarni qo'shimcha yuklash boshqa operatsiyalarni yuklashdan farq qilmaydi, Postfiks shakldagi ++va -- amallarini qayta yuklaganda yana bir int tipidagi parametr kiritilishi kerak.Agar qo'shimcha yuklash uchun global funktsiya ishlatilsa uning birinchi parametri shif tipiga, ikkkinchi parametri int tipiga ega bo'lishi kerak.

Dasturda postfiks ifoda ishlatilganda butun parametr ham qiymatga ega bo'ladi ;

Qo'yidagi dasturda prefiks ++va --hamda postfiks ++va -- operatsiyalarini qo'shimcha yuklash ko'rsatilgan.

```
        # include<iosrtream.h>class pair int N;
          Double x;
          Friend pair & operator ++(pair&);
          Friend pair& operatpr ++(pair,Int);
          Pullic;
          Pair lintn, donblexn
          {N=n; x=xn;}
          Void delplay C'
          {Cout+<c"/n Korodinatalar: N="<<N<c"*x>"<<x;}
Pair& operator--C
{N/=10; x/=10; return*thes;}
Pair& operator --(int k)
{N/=2; X/=2.0; return*this;}
Pair& operator++(pair& p)
P.N*=10; P.x*=10;
Return P;
}
Pair& operator ++(pair& P, int k)
{P.N=P.N*2+k;
P.x=P.x*2+k;
Return P;}
Void mein ()
{Pair Z (10,20,0)
Z.display ();
++ 2;
Z.display ();
--Z;
 Z.display ();
Z ++
Z.display ();
Z --;
Z.display ();
}
```

Dastur bajarilishi natijalari:

```
Koordinatalar:    N=10 X=20
Koordinatadar:    N=100 X=200
Koordinatalar:    N= 10 X=20
Koordinatalar:    N=20 X=40
Koordinatalar:    N=10 X=20
```

Bu misolda prefiks ++ qiymatni 10 marta oshirishni postfiks ++ bo’lsa 2 marta oshirishni bildiradi. Prefiks -- qiymatni 10 marta kamaytirish, postfiks – bo’lsa qiymatni 20 marta kamaytirishni bildiradi.

75 - DARS. LOKAL SINFLAR (STT)

Sinf blok ichida , masalan funktsiya tomonida tariflanishi mumkin. Bunday sinf model' sinf deb ataladi . Lokal sinf kampanentalariga shu sinf tariflangan blok yoki funktsiya tashqarisida murojaat qilish mumkin emas. Lokal sinf statik komponentlarga ega bo'lishi mumkin emas. Lokal sinf ichida shu sinf aniqlangan soniga tegishli nomlari ; statik(statik) o'zgaruvchilar; tashqi (entern) o'zgaruvchilar va tashqi funktsiyalardan foydalanish mumkin. Aftomatik hotira tuzishga tegishli o'zgaruvchilardan foydalanish mumkin emas. Lokal sinflar komponent funktsiyalari faqat joylashinuvchi (line) funktsiya bo'lishi mumkin.

Lokal sinflardan foydalanish hususiyatlarini tushunturish uchun qo'yidagi masalani ko'rib chiqamiz. «Kvadrat» sinfini aniqlash kerak bo'lsin. Kvadrat tamonlarini koordinatalar uqiga parallel deb qaraymiz. Har bir kvadrat berilganligi sifatida markaz koordinatalari va tamon uzunligi olinadi.Kvadrat sinfi ichida «kesma lokal»sinfini aniqlaymiz. Har bir kesmani berilganlari sifatida uchlarining koordinatalarini olamiz. Uchlari mos ravishda olingan to'rtta kesma kvadratni nomi qiladi.Shu usulda kvadrat ekranda tasvirlanadi.

```
1.      #include<conion.h>
#include «point.cpp»
{clarr segment
{point pn, pk;
public:
segment(point pin=point(0,0);
            point  pin.=point(0,0)
{pn.give x()=pin.givex ();
pn.givey()=pin.give y();
pn.give x()=pin.give x();
 pn.give y()=pin.give y();
}
point & beg(void){return pn;}
point & end (void) {return pk;}
void show sey().
{line(pn give x(),pn.givey(),
pk.give x(),pn.give y();};};
segment ab,bc,cd,da;
public
square(point ci=point(0,0),int di=0)
point a,b,c,d;
a.given()=ci.give x()-di/2;
a.give y()=ci.give y()-di/2;
b.give x()=ci.give x()+di/2;
b.give y()=ci.give y()-di/2;
c.give x()=ci.give x()+di/2;
c.give y()=ci.give y()+di/2;
d give x()=ci.give x()-di/2;
d.give y()=ci.give y()+di/2;
ab.bog()=a;ab.end()=b;
bc.bog()=b:bc end()=c;
cd.beg()=c;cd.end()=d;
da.beg()=d;da end()=a;
```

```
}
void show square(void)
{ab.show seg();
bc show seg();
cd.show  seg ();
da.show seg();
}
};
void main()
{int dr=DETECT,mod;
initgraph(&dr,& mod,"c:||borlonde||) bg");
point pi(80,120);
point pr(250,240);
square A(p1,30);
square B(p2,140);
A show square();geych();
B show square();getch();
Closegraph();
}
```

76 - DARS. SINFLAR VA SHABLONLAR.

Shablonli funktsiyalar va sinflar oilasini yaratishga imkon beradi.Shablalar sinfiy yoki parametrlangan tillar deb ham ataladi. Yuqorida ko'rsatilganidek funktsiyalar oilasi shabloni cheksiz ko'p o'zaro yaqin funktsiyalarni aniqlashga imkon beradi va qo'yidagiga ega bo'ladi:

template<shabla parametrlari-ruyhati>f-ya tarifi.

Sinf ta'rifiga tushuvchi kam,butun bir sinflar oilasi nomi bulib hizmat qiladi.Shablon tarifi faqat global bo'lishi mumkin. Shablon kiritilgandan so'ng sinflar obe'ktlari qo'yidagicha ta'riflananadi; parametrlangan sinf nomi <shablan –haqiqiy-parametrlari>obe'kt-nomi(konstruktor parametrlari);

STT tili avtorlariga muvofiq vektorli sinfli ko'rib chiqamiz. Ventar elementlari qanday tipga tegishli bo'lmasin ular ustida bir hil amallar aniqlanishi mumkin. Kutbidagi shablon kerakli hossalarga ega bo'lgan ventorlar sinflarini avtomatik yaratishga imkon beradi.

```
||Template.vec
template<clarr  T>
clarr vector
{T data;
int size;
public;
vector(int);
~Vector(){delete[]data;};
T&aperator[](int i){return data[i];}
};
templote<class T>
vector<t>::Vector(int n)
{data=new t[n];
size=n;
};
```

Qo'yidagi dasturda shu shablan asosida konkret butun sonli va simvolli vektorlarni tushuntirib ko'rib chiqamiz:

```
# include "template.vec"
#include<iortream.h>
main()
{Vecto<int>x(5);
Vector<int>x(5);
For(int I=0;i<5;I++)
{x[i]=I;c[i]='A'+I;}
for(I=0;I<5;I++)
{x[i]=I;c[I]='A'+I;}
for(I=0;I<5;I++)
cont<<" "<<x[I]<<"'<<c[I];
}
```

77 - DARS. SHABLON YARATISH.

Misol uchun massiv sinfi yaratilib bu sinfda massivning summasini va o’rta qiymatini hisoblash usullari mavjud bo’lsin. Agar siz int tipida gi massiv bilan ishlayotgan bo’lsangiz sinf ta'rifi qo’yidagicha bo’lishi mumkin:

```
class array
{
public:
   array(int size);
   long sum(void);
   int average_value(void);
   void show_array(void);
   int add_value(int);
private:
   int *data;
   int size;
   int index;
};
```

Qo’yidagi I_ARRAY.CPP dasturda array sinfidan int tipidagi massivlar bilan ishlash uchun foydalanilgan.

```
#include <iostream.h>
#include <stdlib.h>
class array
{
public:
   array(int size);
   long sum(void);
   int average_value(void);
   void show_array(void);
   int add_value(int) ;
private:
   int *data;
   int size;
   int index;
};
array::array(int size)
{
   data = new int [size];
   if (data == NULL)
   {
      cerr << "Nedostatochno pamyati - programma zavershaetsya " << endl;
      exit(l);
   }
   array:: size = size;
   array::index = 0;
}
long array::sum(void)
{
   long sum = 0;
   for (int i = 0; i < index; i++) sum += data[i];
```

```
  return(sum);
}
int array::average_value(void)
{
  long sum = 0;
  for (int i = 0; i < index; i++) sum += data[i];
  return (sum / index);
}
void array::show_array(void)
{
  for (int i = 0; i < index; i++) cout << data[i] << ' ';
  cout << endl;
}
int array::add_value(int value)
{
  if (index == size) return(-1); // massiv polon
  else
  {
    data[index] = value;
    index++;
    return(0); // uspeshno
  }
}
void main(void)
{
  array numbers (100); // massiv iz 100 el-tov
  int i;
  for (i = 0; i < 50; i++) numbers.add_value(i);
  numbers.show_array();
  cout << "Summa chisel ravna " << numbers.sum () << endl;
  cout << "Srednee znachenie ravno " << numbers.average_value() << endl;
}
```

Dasturda avval massiv 100 elementi taqsimlanadi. So'ngra massivga 50 qiymat add_value. Usuli yordamida yoziladi. Qo'yidagi **array** sinfida **index** o'zgaruvchisida massivda saqlanuvchi elementlar soni yoziladi. Agar massiv sig'dirishi mumkin bo'lgan elementlar sonidan ortiq elemetlar yozishga urinilsa add_value funktsiyasi hato haqida ma'lumot qaytaradi. Ko'rinib turibdiki average_value funktsiyasi index o'zgaruvchisidan massiv elementlari soni o'rta qiymatini aniqlashda foydalaniladi. Dastur new operatoridan foydalanilgan.

Agar dasturda haqiqiy qiymatli massivlar binoan ishlash zarur bulsa Yangi sinf yaratishga to'g'ri keladi. Sinflarni kupaytirmaslik uchun sinflar shablonlaridan foydalanish mumkin. Qo'yida umumiy **array** sinfi shabloni keltirilgagn:

```
template<class T, class T1> class array
{
public:
  array(int size);
  T1 sum (void);
  T average_value(void);
  void show_array(void);
  int add_value(T);
```

```
private:
   T *data;
   int size;
   int index;
};
```

78 - DARS. SINFLARDA VORISLIK VA POLIMORFIZM.

Sinflarda vorislik

Har hil sinflar ob'ektlar va sinflarning uzlari vorislik munosabatlarida bo'lishi mumkin. Bu munosabat obe'ktlar va sinflar ierarhiyasi hosil bo'lishiga olib keladi. Sinflar ierarhiyasi mavjud sinflar yordamida, ya'ngi sinf yaratishga imkon beradi. Mavjud sinflar asosiy (yoki yaratuvchi) bu sinflar asosida shakillangan sinflar hosilaviy (yoki yaratilgan), ba'zida sinf vorislari yoki meroshurlari deb ataladi.
Hosilaviy sinflar asosiy sinflarning ma'lumotlari va metodlarini merosga oladilar. Bundan tashqari ularning o'z ma'lumotlari va metodlari bo'lishi mumkin.
Me'roslik o'tuvchi pommosontalar hosilaviy sinflarga ko'chmaydi, balki asosiy sinflarda qoladi. Agar ahborotlarni qayta ishlash uchun hosilaviy sinfda yuq bo'lgan ma'lumotlar kerak bo'lsa ular avtomatik ravishda asosiy sinfda qidiriladi.

Sodda vorislik

Vorislik hosilaviy sinfning mavjud asosiy sinfning harakteristikalarini merosga olish hususiyatidir. Misol uchun asosiy sinf employee mavjud bo'lsin:

```
class employee
{
public:
   employee(char *, char *, float);
   void show_employee(void);
private:
   char name[64];
   char position[64];
   float salary;
};
```

Dasturda yangi manager sinfi yaratib employee sinfining qo'yidagi elementlarini merosga olishi lozim bo'lsin:

```
float annual_bonus;
char company_car[64];
int stock_options;
```

U holda manager sinfi qo'yidagicha ta'riflanadi:

```
class manager : public employee
{
public:
   manager(char *, char *, char *, float, float, int);
   void show_manager(void);
private:
   float annual_bonus;
   char company_car[64];
   int stock_options;
};
```

Hosilaviy sinf asosiy sinfning elementlariga to'g'ridan – to'g'ri nuqta operatori orqali murojaat qila olmaydi. Qo'yidagi MGR_EMP.CPP dasturda vorislikdan foydalanish ko'rsatiladi:

```
#include <iostream.h>
#include <string.h>
class employee
{
```

```
public:
   employee(char *, char *, float);
   void show_employee(void);
private:
   char name [ 64 ];
   char position[64];
   float salary;
};
employee::employee(char *name, char *position,float salary)
{
   strcpy(employee::name, name);
   strcpy(employee::position, position);
   employee::salary = salary;
}
void employee::show_employee(void)
{
   cout << "Imya: " << name << endl;
   cout << "Doljnost': " << position << endl;
   cout << "Oklad: $" << salary << endl;
}
class manager : public employee
{
public:
   manager(char *, char *, char *, float, float, int);
   void show_manager(void);
private:
   float annual_bonus;
   char company_car[64];
   int stock_options;
};
manager::manager(char *name, char *position, char *company_car, float salary, float
bonus, int stock_options) : employee(name, position, salary)
{
   strcpy(manager::company_car, company_car) ;
   manager::annual_bonus = bonus ;
   manager::stock_options = stock_options;
}
void manager::show_manager(void)
{
   show_employee();
   cout << "Mashina firmih: " << company_car << endl;
   cout << "Ejegodnaya premiya: $" << annual_bonus << endl;
   cout << "Fondovihy optsion: " << stock_options << endl;
}
void main(void)
{
   employee worker("Djon Doy", "Programmist", 35000);
   manager boss("Djeyn Doy", "Vitse-prezident ", "Lexus", 50000.0, 5000, 1000);
   worker.show_employee() ;
   boss.show_manager();
```

```
}
```
Bu misolda manager sinfi konstruktoriga e'tibor berish lozimdir. Asosiy sinfning konstruktorini qo'yidagicha chaqirish lozimdir:
```
manager::manager(char *name, char *position, char *company_car, float salary, float bonus, int stock_options) :
employee(name, position, salary) //————————————————— Konstruktor bazovogo klassa
{
strcpy(manager::company_car, company_car);
manager::annual_bonus = bonus;
manager::stock_options = stock_options;
}
```
Yana shunga e'tibor berinki show_manager funktsiyasi show_employee funktsiyasini chaqirishi mumkin, chunki manager sinfi employee sinfining vorisi bo'lgani uchun, umumiy elemenlariga murojaat qilishi mumkindir. Ikkinchi misol
Misol uchsun book asosiy sinfi mavjud:
```
class book
{
public:
  book (char *, char *, int);
  void show_book(void);
private:
  char title[64];
  char author[b 4];
  int pages;
};
```
Yangi library_card sinfi book sinfiga qo'yidagi elemenlarni qo'shishi lozim:
```
char catalog[64];
int checked_out; // 1, agar tekshirilgan bulsa, aks holda O
```
Dasturda bu sinf vorislik yordamida qo'yidagicha ta'riflanishmi lozim
```
class library_card : public book
{
public:
  library_card(char *, char *, int, char *, int);
  void show_card(void);
private:
  char catalog[64] ;
  int checked_out;
};
```
Qo'yidagi BOOKCARD.CPP dasturida bu sinflardan foydalanish ko'rsatilgan:
```
#include <iostream.h>
#include <string.h>
class book
{
public:
  book(char *, char *, int);
  void show_book(void);
private:
  char title [64];
  char author[64];
```

```
  int pages;
};
book::book(char •title, char *author, int pages)
{
  strcpy(book::title, title);
  strcpy(book::author, author);
  book::pages = pages;
}
void book::show_book(void)
{
  cout << "Nazvanie: " << title << endl;
  cout << "Avtor: " << author << endl;
  cout << "Stranits: " << pages << endl;
}
class library_card : public book
{
public:
  library_card(char *, char *, int, char *, int);
  void show_card(void) ;
private:
  char catalog[64];
  int checked_out;
};
library_card::library_card(char *title, char *author, int pages, char *catalog, int
checked_out) : book(title, author, pages)
{
  strcpy(library_card::catalog, catalog) ;
  library_card::checked_out = checked_out;
}
void 1ibrary_card::show_card(void)
{
  show_book() ;
  cout << "Katalog: " << catalog << endl;
  if (checked_out) cout << "Status: proverena" << endl;
  else cout << "Status: svobodna" << endl;
}
void main(void)
{
  library_card card( "Uchimsya programmirovat' na yazihke C++", "Jamsa", 272,
"101SRR", 1);
  card.show_card();
}
```

Yana shunga e'tibor berinki library_card konstruktori book sinfi konstruktorini chaqiradi.

Nomlar konfliktini hal kilish

Agar bir sinfdan ikkinchisini hosil qilinsa asosiy va hosilaviy sinflarda elementlar nomlari bir hil bo’lishi mumkin. Bu holda 'hosilaviy sinf ichidagi funktsiyalarda hosilaviy sinf elementaridan foydalaniladi. Misol uchun book i library_card sinflari price

elementlaridan foydalansin. Agar aniq ko’rsatilmagan bo’lsa library_card chsinfi funktsiyalari shu sinfning price elementidan foydalanadi. Agar library_card sinfi funktsiyalaori book sinfi price elementiga murojaat qilishi lozim bo’lsa ruhsat operatoridan foydalanishi lozim, masalan book::price. Agar show_card funktsiyasi ikkala narhni ekranga chiqarishi lozim bo’lsa qo’yidagi operatorlardan foydalanishi lozim:

```
cout << "Bibliotechnaya tsena: $" << price << endl;
cout << "Prodajnaya tsena: $" << book::price << endl;
```

79 - DARS.VORISLIKDA MUROJAAT HUQUQLARI.

Vorislikda asosiy sinfning ba'zi komponenta ma'lumotlari yoki komponenta funktsiyalari hosilaviy sinfda yangidan ta'riflanishi mumkin. Bu holda asosiy sinfning komponentalariga hosilaviy sinfdan to'g'ridan-to'g'ri murojaat qilib bo'lmaydi. Bu holda ko'rinish doirasini aniqlovchi ^:: amalidan foydalanish lozimdir. Har qanday komplament sinf o'z urnida boshqa sinflar uchun hosilaviy bo'ladi. Sinflar va ob'ektlar ierarhiyasida hosilaviy ob'ekt hamma asosiy sinflarning ruhsat berilgan komponentalarini hisobga oladi. Sinflar volisligida komponentalarning murojaat huquqlari katta rol' o'ynaydi. Haqiqiy sinf uchun komponentalari ta'sir doirasida yotadi. Shuning uchun sinfga tegishli har qanday funktsiya ihtiyoriy komponenta ma'lumotlariga murojaat qilishi va sinfga tegishli ihtiyoriy funktsiyani chaqirishi mumkin. Sinf tashqarisida faqat public huquqiga ega komponentalarga murojaat qilish mumkin. Sinflar perergiyasida sinf komponentalariga murojaat huquqlari qo'yidagilar:

Hususiy (private) metodlar va ma'lumotlarga faqat sinf ichida murojaat qilish mumkin.

Himoyalangan (protected) komponentalari o'z sinflari va shu sinfga me'roshur bo'lgan hamma hosilaviy sinflarga murojaat qilish mumkin.

Umumiy (public) komponentlar global ya'ni dastur ihtiyoriy nuqtasidan murojaat qilish mumkin.

Himoyalanganlari murojaat sinf clarl, istruct yoki union so'zlarining qaysi biri bilan ta'riflanganligiga ham bog'likdir.

A " ekrandagi nuqta" asosiy sinf hisoblansa uning asosida " ekrandagi darcha sinfni ugirish mumkin. Bu sinf berilganlarni ikki nuqta :

& chap yuqori burchakli aniqlovchi nuqta

& darcha o'lchovlarini ya'ni chap yuqori burchakka nisbatan koordinatalar o'qi buyicha siljish.

Ekrandagi darcha sinfi me'todlari:

& darchani H o'qi bo'yicha DX ga surish

& darchani U o'qi bo'yicha DY ga surish

& chap yuqori burchak H koordinatasini aniqlash

& chap yuqori burchak U koordinatasini aniqlash;

& H o'qi buyicha darcha uzunligini aniqlash

& Y uki buyicha darcha uzunligini aniklash

Darchaning ekrandagi konstrutuktori:

& ekranda chap yuqori burchagi va ulchamlari asosida berilgan nomli darcha yaratish;

Ekrandagi destruktori

& berilgan nomli darchani yo'q qilish:

Nasldan o'tuvchi komponentalarga qo'shimcha jpot sinfiga qo'yidagi komponentalarni kiritamiz: tasvir radiusi (rad); ekralus sos etilishi (vir=0 ekranda tasvir yuk; vi1==1 ekranda tasvir bor); tasvirni bitli matnda saylash chun ajratilgan hotira qismiga ko'rsatgich pspot.

Spot.cpp

```
# Ifudef. Spot
 # Spot1
"Include " " point. Epp"
 Clall spot;
 {Int rad;
 Int vil;
```

```
   Int tag;
   Void * pspot;
   Public;
   Spot (int xi, int yi, int ri);
   Point (xi, yi)}
   {int size ;
   vir =0;tag=0;rad=ri;
   Size=image size (xi-ri; yi-ri; xis ri; yiri);
   Pspot=neo char [Size];
   }
    ~ Spot ()
    {h del();
     tag =0;
    Delete pspot;
    }
  Voit show ()
   {If (tag==0)
   {Cirele (x, y, rad);
    Flood siell (x, y, getcolor ());
   Get image (x-rad, y-rad, y+rad, pspot);
   Tag=1};
   Else
   Putimage (x-rad, y-rad, pspot, XOR-PUT);
  Vi1=1;
 }
 Void hide ()
 {If (vi1==0) return;
 Putimage (x-rad,y-rad, pspot, XOR-PUT);
Vi1=0;
}
Void move (int xn, int yn)
{Hide ();
 x- xn, y-yn;
 Shov ();
 }
Viod vary (float dr)
{Floata;
 Int size;
 Hide ();
 Tag=0;
 Delete pspot;
A=dr*rad;
If (a<=0) rad=0;
Else rad= (int) a;
Size=imagerize (x-rad; y-rad, x+rad, y+rad);
New char [size];
Show ();
}
 int& giver (void);
{Return rad;}
```

```
};
# Endif
```

Spot sinfida construktor destruktor ~ spot () va beshta metod ko’rsatilgan:

Show ()-- ekranga doirani chizib, bitli tasvirni hotiraga olish;
Hide ()-- ekrandan doira tasvirini uchirish;
Move ()--tasvirni ekranning bitta joyiga ko’chirish;
Vary ()--ekrandagi tasvirni o’zgartirish (kichkinalashtirish yoki kattalashtirish);
Giver () --doira radiusiga murojatni ta'minlash;

Point sinfidan Jpot sinfi naslga nuqta markazi (h,u) koordinatalarini va givek, givey metodlarni oladi, Point : : show () va point : : move () metodini huddi shu nomli yangi funktsiyalar bilan almashtirilgan.point :: hide funktsiyasi nomi o’tmaydi chunki point sinfida u hususiy (private) statiyasiga ega. Stop() konstruktor uch parametrga ega - mernez koordinatalari (xi,yi) va doira radiusi (ri).

Avval point sinfi konstruktori chaqiriladi bu konstruktor xi,yi ga mos keluvchi haqiqiy parametr asosida doira markazini aniqlaydi. Asosiy sinf konstruktori har doim hosilaviy sinf konstruktoridan oldin chaqiriladi. So’ngra spot() sinfi konstruktolari boshlanadi. Bu konstruktor vi1, tag parametrlarining boshlang’ich qiymatini aniqlaydi va ri gamos keluvchi haqiqiy parametr qiymati asosida doira radiusi red aniqlanadi. Standart funktsiya imagelizi yordamida doira joylashuvchi kvadratik operativ hotirada aniqlash uchun zarur bo’lgan hotira hajmi hisoblanadi. Kerakli hotira new standart operatsiya yordamida ajratib size elimentidan iborat chur massivlar yoziladi. Agar aytilgan hotira spot sinfida protected statutisiga ega bo’lgan spot ko’rsatkichiga ulanadi.

80 - DARS. VORISLIKDA DESTRUKTORLAR HOSSALARI.

Sinfning har bir ob'ekti yaratilganda sinf konstruktori chaqirilib, ob'ekt uchun kerakli hotira yaratish va liniyalizatsiya qilish vazifalarini bajaradi. Ob'ekt yuqotilganda yoki sinf ta'sir doirasidan tashqariga chiqilganda teskari inertsiyalarni boshqarish kerak bo'lib,bu lediatsiyalar ichida eng kirishli hotirani ozod qilishdir. Bu vazifalarni boshqarish uchun sinfga mahsus funktsiya dastruktor kiritiladi. Dastruktor quyidagi shaklga ega bo'lgan aniq nomga ega ~ sinf-nomi.

Dastruktor hatto void tipidagi parametrlarga ega bo'lmaydi va hatto void tipidagi qiymat qaytarmaydi. Destruktor statusi ikki e'lon qilinmagan bo'lsa umumiydir. Sodda sinflarda dastruktor avtomatik aniqlanadi, misol uchun paint sinfida destruktor e'lon qilinmagan va pomilyator qo'yidagi dasturlarni avtomatik chaqiradi

```
Point () {};
```

Spot sinfida destruktor anik kurinishga ega;

```
Spot () {hide (); tag=0;delete [] p1pot;}
```

Bu dastruktor vazifalari doira tasvirini spot::hide() funktsiyasi orqali o'chirish; tag belgisiga 0 qiymatini berish; ob'ekt bitli tasvirni saqlash uchun ajratilgan hotirani tozalash.

Destruktorlar naslga o'tmaydi, shuning uchun hosilaviy sinfda destruktor mavjud bo'lmasa asosiy sinfdagi destruktor chaqirilmaydi. Balki kominator tomonidan yaratiladi. Ko'rilayotgan misolda qo'yidagicha;

```
Public: ~spot () {~point ();}
```

Asosiy sinflar destruktorlar ruyhatda ko'rsatilganidek teskari tartibda boshariladi. Shunday qilib ob'ektlarni o'chirish tartibi yaratilish tartibiga teskaridir.Sinf ob'ektini va asosiy sinflar uchun destruktorlar avtomatik chaqiriladi. Agar ob'ekt yaratilganda dasturda hotira ajratilgan bo'lsa destruktor dasturda chaqirilishi lozim. Spot sinfi ob'ektlari bilan ishlovchi dasturni keltiramiz:

```
# Include<graphics.h>
# Include<conio. H >
# Include "spot. Cpp"
 Void main ()
{Int dr=DETECT, mod;
Initgraph (8dr, 8mod);
{Spot A (200,50,20);
Spot D (500,200,30);
A.      show;
Det ch ();
O. Show ();
Det ch; A. Move (50,60);
Det ch (); {closegrap ();}
```

ELIPS, SPOT va SPOTELLI sinflarida POINT sinfining x,y komponentalari nuqtasining ekrandagi koordinatalari naslga o'tadi.POIN sinfida ular himoyalangan (protected) sifatida aniqlangan va bu statusti hosilaviy sinflarda ham saqlab qoladi. SPOTELLI sinfi konstruktori hech qanday vazifa bajarmaydi ketma-ket ELLIPS sinfi va SPOT sinfi konstruktorlari chaqiriladi.U holda yaratilayotgan shakillar markazlari ustma-ust tushadi, doira radiusi sifatida ellipsning radiusi olinadi. Bu holda ishlatiladigan min() funktsiyasi hususiy (private) joylashtirilgan (inline) funktsiya sifatida aniqlangan. Bevosita bo'lmagan sinflar ob'ektlarning bir necha marta yaratilishi oldini olish uchun, bu asosiy sinf virtual sinf deb e'lon qilinadi. Masalan qo'yidagi ta'rifda H sinfi virtual bo'ladi:

```
Class base
{int  j=0,char c='*' )
{jj=j;
 cc=c;
 }
 };
double dd;
public:
dbase (double d=0.0):base()
{dd=d;}
 };
 class j base ; public virtual base
 {float ff ;
 public ;
 jbase (float j=0.0):base ()
 {ff=f;}
 };
 class top:public dbase,public jbase
 {long tt;
  public;
  top(long t=0):dbase (),jbase ()
  {tt=t;}
   };
   void main()
  {cont<<"\n asosiy sinf:size of (base) ="<c  h zeaf (base )
   cont << "\n tugri asos :size of (d base)="<<size of(dbase)
   cont << :\ n tugri asos :size of (fbase)="<<size of (fbase)
   cont <<\n hosilaviy sinf :sizeof (top)="<<sizeof (top);
```

ko'plab vorislikda bitta asosiy sinf hosilaviy virtual yoki no virtual bir necha marta kirishi mumkin. Misol keltiramiz:

```
clarr x{.....};
clarr'/:virtual public x{....}:
clarr z:virtual public x{...};
clarr b :virtual public x{...};
clarr c:virtual public x{....};
clarr e:public x{....};
clarr d:public x{.....};
clarr a :public p;public b;
          public y ,public z ;
           public c ,public e{.....};
```

Bu misolda A sinf ob'ekti o'z ichiga H sinfining 3 ob'ektini bitta virtual B ,Y,C,Z sinflari bilan birgalikda ishlatuvchi ; va ikki virtual mos ravishda D va E sinflariga tegishli bulgan.Shunday kilib virtuallik bu sinfning hususiyati emas, balki nasldan-nalga utishning hususiyatidir.

Nasldorlik va ko'plik nasldorlikdan foydalanilganda har hil sinflarning bir hil nomli kompanentalariga murojaat qilganda har hil talqin yuzaga keladi. Bu har hil talqin oldini olishning eng sodda va ishonchli usuli --kompanentalarining kvalifikatsiyalangan nomlaridan foydalanishdir. Kompanenta nomini kvalifikatsiya qilish uchun sinf nomi ishlatiladi. Qo'yidagi misollar kvalifikatsiyalangan nomlardan foydalanish ko'rsatilgan.

```
Clarr x {public:int d;...};
```

```
Clarr y {public:int d;...};
Clarr z :public y,public y,
          {public ;
           int d;
          --------
          d=y::d+y::d;
           --------
           };
```

Bu dasturning boshqa ko'rilgan dasturlardan printsipial farqi spot sinfida HIDE() kompanenti funktsiyasiga murojat qiluvchi dastruktor ishlashi bilan bog'lik, ishlab chiqarishni blokning mavjudligidir. Agar dasturli point sinfli dasturlar kabi ichki bloksiz yaratilsa dastruktor ochik murojat qilinmaganda faqat dastur tugaganda ya'ni grafik rejim berkitilganda chaqiriladi. Bu hatoni ikki yul bilan oldini olish mumkin. Yoki A va D ob'ektlani yuqotish uchun destruktorni dasturda chaqirish va shundan so'ng grafik rejimni berkitish yoki grafik rejimni initsializatsiya qilgandan so'ng A va D ob'ektlar aniqlangan ichki blok kiriting.

Bu ichki blokdan chiqilayotganda A va D ob'ektlar avtomatik uchiriladilar.Buning uchun dastruktor ikki marta avtomatik chaqiriladi.Grafik rejim A,D ob'ektlar yuqotilgandan sung tashqi blokda berkitiladi.

Qo'yidagi misolda ikkinchi usul qo'llanilishi ko'rsatilgan:

```
-----
getch ();d. Vary (3);
getch();
A.spot ::<spot();
Getch();
D.spot::<spot ();
Closegraph();
}
```

Sodda misol

Misol uchun computer_screen sinfi mavjud bo'lsin:

```
class computer_screen
{
public:
  computer_screen(char *, long, int, int);
  void show_screen(void);
private:
  char type[32] ;
  long colors;
  int x_resolution;
  int y_resolution;
};
```

Bundan tashqari mother_board sinfi ham mavjud bo'lsin:

```
class mother_board
{
public:
  mother_board(int, int, int);
  void show_mother_board(void);
private:
  int processor;
  int speed;
```

```
  int RAM;
};
```
Bu sinflardan foydalanilgan holda yangi computer sinfini yaratish mumkin:
```
class computer : public computer_screen, public mother_board
{
public:
  computer(char *, int, float, char *, long, int, int, int, int);
  void show_computer(void);
private:
  char name[64];
  int hard_disk;
  float floppy;
};
```
Bu sinf uzining avlod sinflarini sinf nomidan so'ng ko'rsatadir.
```
class computer : public computer_screen, public mother_board //————————>
```
Qo'yidagi COMPUTER. CPP dasturida computer_screen va mother_board sinflari asosida computer sinfi yaratiladi:
```
#include <iostream.h>
#include <string.h>
class computer_screen
{
public:
  computer_screen(char *, long, int, int);
  void show_screen(void);
private:
  char type[32];
  long colors;
  int x_resolution;
  int y_resolution;
};
computer_screen::computer_screen(char *type, long colors, int x_res, int y_ree)
{
  strcpy(computer_screen::type, type);
  computer_screen::colors = colors;
  computer_screen::x_resolution = x_res;
  computer_screen::y_resolution = y_res;
}
void computer_screen::show_screen(void)
{
  cout << "Tip ekrana: " << type << endl;
  cout << "Tsvetov: " << colors << endl;
  cout << "Razreshenie: " << x_resolution << " na " << y_resolution << endl;
}
class mother_board
{
public:
  mother_board(int, int, int);
  void show_mother_board(void);
private:
  int processor;
```

```
  int speed;
  int RAM;
};
mother_board::mother_board(int processor, int speed, int RAM)
{
  mother_board::processor = processor;
  mother_board::speed = speed;
  mother_board::RAM = ram;
}
void mother_board::show_mother_board(void)
{
  cout << "Protsessor: " << processor << endl;
  cout << "Chastota: " << speed << "MGts" << endl;
  cout << "OZU: " << RAM << " MVayt" << endl;
}
class computer : public computer_screen, public mother_board
{
public:
  computer(char *, int, float, char *, long, int, int, int, int, int);
  void show_computerf void);
private:
  char name [64];
  int hard_disk;
  float floppy;
};
computer::computer(char *name, int hard_disk, float floppy, char *screen, long colors,
int x_res, int y_res, int processor, int speed, int RAM) : computer_screen(screen, colors,
x_res, y_res), mother_board(processor, speed, ram)
{
  strcpy(computer::name, name);
  computer::hard_disk = hard_disk;
  computer::floppy = floppy;
}
void computer::show_computer(void)
{
  cout << "Tip: " << name << endl;
  cout << "Jestkiy disk: " << hard_disk << "MVayt" << endl;
  cout << "Gibkiy disk: " << floppy << "MVayt" << endl;
  show_mother_board();
  show_screen();
}
void main(void)
{
  computer my_pc("Compaq", 212, 1.44, "SVGA", 16000000, 640, 480, 486, 66, 8);
  my_pc.show_computer();
}
```

Bu misolda computer sinfi konstruktori, mother_board va computer_screen konstruktorlarini chaqiradi: computer::computer(char *name, int hard_disk, float floppy, char *screen, long colors, int x_res, int y_res, int processor, int speed, int RAM) : computer_screen(screen, colors, x_res, y_res), mother_board(processor, speed, RAM)

81 - DARS. SINFLAR IERARHIYASINI QO'RISH.

C++ tilida bir sinf uchun ajdod sinf o'z o'rnida boshqa sinfning avlodi bo'lishi mumkin. Misol uchun somputer sinfi workstation sinfi uchun ajdod sinf bo'lsin:

```
class work_station : public computer
{
public:
  work_station (char *operating_system, char *name, int hard_disk, float floppy, char
*screen, long colors, int x_res, int      y_res,int processor, int speed, int RAM);
  void show_work_station(void);
private:
  char operating_system[64];
};
```

Bu misolda workstation sinfi konstruktori computer sinfining konstruktorini chaqiradi u bo'lsa somruter_screen va mother_board sinfi konstruktorlarini chaqiradi:

```
work_station::work_station( char *operating_system, char *name, int hard_disk, float
floppy, char *screen, long colors, int    x_res, int y_res, int processor, int speed, int
RAM) : computer (name, hard_disk, floppy, screen, colors, x_res, y_res,    processor,
speed, RAM)
{
  strcpy(work_station::operating_system, operating_system);
}
```

Bu misolda computer sinfi asosiy sinfdir. Lekin computer sinfi computer_screen va mother_board sinflarining avlodidir. Natijada work_station sinfi hama uch sinf harakteristikalarini merosga oladi.

82 - DARS. POLIMORFIZM.

Polimorf ob'ekt bu dastur bajarilishi davomida shaklini o'zgartirishi mumkin bo'lgan ob'ektdir. Misol uchun telenfon sinfi kiritilgan bo'lsin:

```
class phone
{
public:
   void dial(char "number) { cout << "Nabor nomera " << number << endl; }
   void answer(void) { cout << "Ojidanie otveta" << endl; }
   void hangup(void) { cout << "Zvonok vihpolnen - povesit' trubku" << endl; }
   void ring(void) { cout << "Zvonok, zvonok, zvonok" << endl;)
   phone(char *number) { strcpy(phone::number, number); };
private:
   char number[13];
);
```

Kuyidagi PHONEONE.CPP dasturi phone sinfidan foydalanadi:

```
#include <iostream.h>
#include <string.h>
class phone
{
public:
   void dial(char *number) { cout << "Nabor nomera " << number << endl; }
   void answer(void) { cout << "Ojidanie otveta" << endl; }
   void hangup(void) { cout << "Zvonok vihpolnen - povesit' trubku" << endl; }
   void ring(void) { cout << "Zvonok, zvonok, zvonok" << endl; }
   phone(char *number) { strcpy(phone::number, number); };
private:
   char number[13];
};
void main(void)
{
   phone telephone("555-1212");
   telephone.dial("212-555-1212");
}
```

Agar tugmali va diskli telefon yaratish lozim bo'lsa va qo'ng'iroq qilish uchun 25 tsent to'lash lozim bo'lsa vorislik yordamida touch_tone va pay_phone schinflarini yaratish mumkin:

```
class touch_tone : phone
{
public:
   void dial(char * number) { cout << "Pik pik Nabor nomera " << number << endl; }
   touch_tone(char *number) : phone(number) { }
};
class pay_phone : phone
{
public:
   void dial(char * number)
   {
      cout << "Pojaluysta, oplatite " << amount << " tsentov" << endl;
      cout << "Nabor nomera " << number << endl;
```

```
  }
  pay_phone(char *number, int amount) : phone(number) { pay_phone::amount =
amount; }
private:
  int amount;
};
```

Yangi touch_tone va pay__phone sinflari hususiy dial usulidan foydalanadi. Qo'yidagi NEWPHONE.CPP dasturida shu sinflardan foydalanilgan

```
#include <iostream.h>
#include <string.h>
class phone
{
public:
  void dial(char *number) { cout << "Nabor nomera " << number << endl; }
  void answer(void) { cout << "Ojidanie otveta" << endl; }
  void hangup(void) { cout << "Zvonok vihpolnen - povesit' trubku" << endl; }
  void ring(void) { cout << "Zvonok, zvonok, zvonok" << endl; }
  phone(char *number) { strcpy(phone::number, number); };
protected:
  char number[13];
);
class touch_tone : phone
{
public:
  void dial(char *number) { cout << "Pik pik Nabor nomera " << number << endl; }
  touch_tone(char *number) : phone(number) { }
};
class pay_phone : phone
{
public:
  void dial(char * number) { cout << "Pojaluysta, oplatite " << amount << " tsentov"
<< endl; cout << "Nabor nomera       " << number << endl; }
  pay_phone(char * number, int amount) : phone(number) { pay_phone::amount =
amount; }
private:
  int amount ;
};
void main (void)
{
  phone rotary("303-555-1212");
  rotary.dial("602-555-1212");
  touch_tone telephone("555-1212");
  telephone.dial("212-555-1212");
  pay_phone city_phone("555-1111", 25);
  city_phone.dial("212-555-1212");
}
```

Dastur bajarilganda ekranga qo'yidagi ma'lumotlar chiqadi:

```
S:\> NEWPHONE <Enter>
Nabor nomera 602-555-1212
Pik pik Nabor nomera 212-555-1212
```

Pojaluysta, oplatite 25 tsentov
Nabor nomera 212-555-1212
Bu misolda polimorf ob'ektlarpdan foydalanilmagan.

POLIMORF OB'EKT-TELEFON YARATISH

Bir qo'ng'iroqdan ikkinchisiga telefon o'z shaklini o'zgartirishi lozim bo'lsin. Polimorf ob'ekt yaratish uchun avval virtual suzi yordamida virtual usullar yaratiladi:

```
class phone
{
public:
   virtual void dial(char •number) { cout << "Nabor nomera " << number << endl; }
   void answer(void) { cout << "Ojidanie otveta" << endl; }
   void hangup(void) { cout << "Zvonok vihpolnen - povesit' trubku" << endl; }
   void ring(void) { cout << "Zvonok, zvonok, zvonok" << endl; }
   phone(char *number) { strcpy(phone::number, number); };
protected:
   char number[13];
};
```

So'ngra dasturda asosiy sinf ob'ektiga ko'rsatkich yaratasiz.

```
phone *poly_phone;
```

Shaklni o'zgartirish uchun bu ko'rsatkiyach qiymatiniga hosilaviy sinf ob'ekti adresiga teng qilinadi:

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

Bu misolda (phone *), tiplarni keltirish operatoridir.
Qo'yidagi POLYMORP.CPP dasturda shu usuldan foydalanilgandir:

```
#include <iostream.h>
#include <string.h>
class phone
{
public:
   virtual void dial(char *number) { cout << "Nabor nomera " << number << endl; }
   void answer(void) { cout << "Ojidanie otveta" << endl; }
   void hangup(void) { cout << "Zvonok vihpolnen - povesit' trubku" << endl; }
   void ring(void) { cout << "Zvonok, zvonok, zvonok" << endl; }
   phone(char *number) { strcpy(phone::number, number); };
protected:
   char number[13] ;
};
class touch_tone : phone
{
public:
   void dial(char * number) { cout << "Pik pik Nabor nomera " << number << endl; }
   touch_tone(char *number) : phone(number) { }
};
class pay_phone: phone
{
public:
   void dial(char *number) { cout << "Pojaluysta, oplatite " << amount << " tsentov"
<< endl; cout << "Nabor nomera "       << number << endl; }
```

```
   pay_phone(char *number, int amount) : phone(number) { pay_phone::amount =
amount; }
private:
   int amount;
};
void main(void)
{
   pay_phone city_phone("702-555-1212", 25);
   touch_tone home_phone("555-1212");
   phone rotary("201-555-1212") ;
      // Sdelat' ob'ekt diskovihm telefonom
   phone *poly_phone = &rotary;
   poly_phone->dial("818-555-1212");
      // Zamenit' formu ob'ekta na knopochnihy telefon
   poly_phone = (phone *) &home_phone;
   poly_phone->dial("303-555-1212");
      // Zamenit' formu ob'ekta na platnihy telefon
   poly_phone = (phone *) &city_phone;
   poly_phone->dial("212-555-1212");
}
```

Dastur bajarilishi natijasida ekranga kuyidagi ma'lumotlar hosil buladi:

```
S:\> POLYMORP <ENTER>
Nabor nomera 818-555-1212
Pik pik Nabor nomera 303-555-1212
Pojaluysta, oplatite 25 tsentov
Nabor nomera 212-555-1212
```

83 - DARS. G'AYRI ODDIY HOLATLARNI DASTURLASH.

G'ayri oddiy holatlarga nol'ga bo'lish, fayl ohiri kabi holatlar kiradi. G'ayri oddiy holatlarni dasturlash uchun C++ tilida Quyidagi uchta hizmatchi so'z ishlatiladi:
try (nazorat qilish)
catch (ilib olish)
throw (otli, hosil qilish, generatsiya qilish)
try – hizmatchi so'zi dastur matni ihtiyoriy qismida nazorat qiluvchi blok tashkil qilishga imkon beradi;
try generatorlar
try generatorlar ichida ta'riflar e'lonlar va oddiy generatorlardan tashkil topadi. G'ayri oddiy hodisalar hosil qiluvchi qo'yidagi operator ham ishlatiladi:
throw ifoda.
Bunday operator bajarilganda mahsus chetlanish deb ataluvchi statik ob'ekt hosil qilinadi. Bu obekt tili ifoda tili orqali aniqlanadi.
chetlanishli qayta ishlovchilar quyidagi ko'rinishga ega bo'ladi.

```
catch (chetlanish tip nomi )

{dasturlar}
```

Figurali qavs ichidagi operatorlar chetlanishlarni qayta ishlash bloki deb ataladi. Chetlanishliklarni qayta ishlovchi tashqi tomondan va ma'no jihatdan qiymat qaytarmaydigan bitta parametrli funktsiyaga yaqindir. Agar qayta ishlovchilar bir nechta bo'lsa, ular chetlanish tillari farq qilishlari lozimdir. Chetlanishlarni dasturlashni Evklid algoritm misolida ko'rib chiqamiz. Bu algoritm ikki butun manfiy bo'lmagan sonlarning EKUBini topishga mo'ljallangandir. Algoritm har bir qadamida quyidagi amallar bajariladi.
Agar x>=y bo'lsa javob topilgan
Agar x<y bo'lsa y=y-x
Agar x>y bo'lsa x=x-y
Quyidagi dastur GCM ( ) funktsiyasini o'z ichiga olib, bu funktsiya nazorat qiluvchi blokni o'z ichiga oladi:

```
#include <iostream.h>
int GCM ( int x, int y )
{
try { if (x==0)(y==0)
throw " \ n zero!";
if (x<0) throw  " \ negative parameter1";
If (y<0) throw " \ negative parameter 2";
While (x:=y);
else
y=y-x
}
return x;
}
catch ( count char* report)
{cer2 << report << "x= "<<x<<", y="<< y;
return 0
```

```
}
}
void main ( )
{
count << " \ n GCM (66,44)= "" " \ n GCM (66,44);
count << " \ n GCM (0,7)= "" " \ n GCM (0,7);
catch (con 2t char * report)
{cer2<<report;}
}
```

Natija:

GCM_New (66,44)=22
zero: x=0, y=7 GCM

GCM (0,7)=0
negdtive parametr1. x=-12, y=8
GCM (-12,8)=0

Bu dastur birinchi chetlashishni qayta ishlagandan so'ng ishni to'htadi. Chetlanish funktsiya tanasidan tashqaridan qayta ishlagani uchun, qayta ishlovchi funktsiya parametriga murojat qila olmaydi. Bu kamchilikdan halos bo'lish uchun cheklanishni mahsus sinov ob'ekti sifatida hosil qilish mumkindir.
Quyidagi misolda DATA sinfi kiritilgandir.

```
# include <iostream>
struct DATA
{ int n,m;
char*S;
DATA (int x, int y, char*c)
{n=x; m=y; s=c;)
};
int GCM_ONE (int n, int y);
{if (x==0||y==0) throw DATA (x,y,"\nZERO!");
if(x<0) throw DATA (x,y "\n Negative parametr1");
if(y<0) throw DATA (x,y "\n Negative parametr2");
while (n!=y)
{if(n>y) x=n-y;
else y=y-x;
}
return x;
}
void main ( )
{ try
{count<c"\nGCM(66,44)="<<GCM_ONE(66,44);
count<c"/nGCM_ONE(0,7)="<<GCM_ONE(0,7);
count<c"/nGCM_ONE(-12,8)="<<GCM_ONE(-12,8);
}
catch (DATA d);
{cerr<<d.S<<"x="<<d.n<<";y="<<d.m}
}
```

Natija
GCM_ONE(66,44)=22
ZERO!x=0,y=7

DATA ishora ob'ekti bu misolda funktsiya tanasida sinf bajarilganda yaratiladi. Bu ob'ekt cheklanish bo'lmaganida chaqirish nuqtasida bo'lar edi. Shunday qilib chegaranishlar sinf ob'ekti bo'lishi lozimdir. Bu sinflarni global ta'riflash shart emas. Asosiy talab tallanish nuqtasida ma'lum
(catch)
misolida DATA sinfi ichida GCM_TWO ( ) funktsyasi. Lekin cheklanishlarni qayta ishlash to'g'ri amalga oshiriladi.

```
# include <iostream>
int GCM_ONE (int x, int y)
{struct DATA
{int n, m;
char8S;
DATA (int x, int y, char*c)
{n=x; m=y; s=c;}
};
if (x==0||y=0) throw DATA (x,y "\ZERO!");
if (x<0) throw DATA (x,y, "\n Negative parametr1");
if (y<0) throw DATA (x,y, "\n Negative parametr2");
while (x!=y)
{if (x>y) x=x-y;
else y=y-x;
}
return x;
}
void main ( )
{struct DATA
{int n, m;
char*S;
DATA (int n, int y, char*c)
{n=x; m=y; S=c;)
};
try
{count<c"\nGCM(66,44)="<<GCM_ONE(66,44)=21
<<GCM_ONE(66,44);
cont <c "\nGCM_ONE (-12,8)=11
GCM_ONE (-12,8);
Cont <c"\n GCM_ONE(0,7)="<<GCM_ONE (0,7)
}
catch (DATA d)
{cerr <<d.S"x>" <<d.n<<",y="
<<d.m;}
}
```

Dastur natijasi:
GCM_TWO (66,44)=22
Negative parametr1
X=12, y=8

G’ayri oddiy holatlar sinflar sifatida

Dasturlarda g’ayri oddiy holatlar sinf sifatida aniqlanadi. Misol uchun qo’yidagi holatlar fayllar Bilan ishlash uchun uchta g’ayrioddiy holatlarni aniqlaydi s faylni:

```
class file_open_error {};
class file_read_error {};
class file_write_error {};
```

G’ayri oddiy holatlarni aniqlash uchun try operatoridan foydalanish lozimdir. Misol uchun qo’yidagi try operatori file_sopy funktsiyasini chaqirishda hosil bo’luvchi g’ayrioddiy holatni aniqlashga ruhsat beradi

```
try
{
  file_copy("SOURCE.THT", "TARGET.THT") ;
};
```

Qaysi holat vujudga kelganligini aniqlash uchun try operatoridan so’ng bir nechta catch operatori joylashtirilishi lozimdir:

```
try
{
  file_copy("SOURCE.THT", "TARGET.THT") ;
};
catch (file_open_error)
{
  cerr << "Oshibka otkrihtiya ishodnogo ili tselevogo fayla" << endl;
  exit(1);
}
catch (file_read_error)
{
  cerr << "Oshibka chteniya ishodnogo fayla" << endl;
  exit(1);
}
catch (file_write_error)
{
  cerr << "Oshibka zapisi tselevogo fayla" << endl;
  exit(1);
}
```

Bu misolda har qanday hato yuz berganda mos ma'lumort chiqarilib dastur o’z ishini to’htatadi. Agar g’ayrioddiy hodisa yuz bermasa catch operatsiyasi ishlatilmaydi.

G’ayri oddiy holatlarni generatsiya qilish

G’ayri oddiy hodisalarni generatsiya qilish uchun throw operatoridan foydalanish lozimdir. Misol uchun

```
void file_copy(char *source, char *target)
{
  char line[256];
  ifstream input_file(source);
  ofstream output_file(target);
  if (input_file.fail())
     throw(file_open_error);
  else
     if (output_file.fail()) throw(file_open_error);
  else
```

```
  {
    while ((! input_file.eof()) && (! input_file.fail()))
    {
      input_file.getline(line, sizeof(line)) ;
      if (! input_file.fail()) output_file << line << endl;
      else throw(file_read_error);
      if (output_file.fail()) throw (file_write_error) ;
    }
  }
}
```

Dasturda ma'lum g'ayri oddiy holatlarni generatsiya qilish uchun throw operatoridan foydalanilgan.

G'ayri oddiy holatni qayta ishlovchini ta'riflash

Dastur g'ayri oddiy holatni generatsiya qilganda kompilyator g'ayri oddiy holatni qayta ishlovchi funktsiyani chaqiradi. Misol uchun qo'yidagi nuke_meltdown gayri oddiy hodisa sinfi bu holatni qayta ishlovchi nuke_meltdown funktsiyasini anilqaydi:

```
class nuke_meltdown
{
public:
  nuke_meltdown(void){ cerr << "\a\a\aRabotayu! Rabotayu! Rabotayu!" << endl; }
};
```

Bu misoldjva dasturda nuke_meltdown g'ayri oddiy hodisasi generatsiya bo'lganda, C++ nuke_meltdown funktsiyasi operatorlarini ishga tushiradi, shundan so'ng g'ayri oddiy holatni aniqlashgga ruhsat beruvchi try operatoridan so'ng kelgan operatorga boshqarishni uzatadi. Qo'yidagi MELTDOWN.CPP dasturi nuke_meltdown funktsiyasini bajarilishini ko'rsatadi. Bu dasturda try operatoridan g'ayri oddiy holatni aniqlashda ruhsat berish uchun foydalaniladi. Shundan so'ng dastur add_u232 funktsiyani amount parametri bilan chaqiradi. Agar bu parametr qiymati 255 dan kam bo'lsa, funktsiya muvaffaqiyatli bajarilmoqda. Agar parametr qiymati 255 dan oshiq bo'lsa funktsiya nuke_meltdown g'ayri oddiy holatini generatsiya qiladi:

```
#include <iostream.h>
class nuke_meltdown
{
public:
  nuke_meltdown(void){ cerr << "\a\a\aRabotayu! Rabotayu! Rabotayu!" << endl; }
};
void add_u232(int amount)
{
  if (amount < 255) cout << "Parametr add_u232 v poryadke" << endl;
  else throw nuke_meltdown();
}
void main(void)
{
  try
  {
    add_u232(255);
  }
  catch (nuke_meltdown)
```

```
  {
    cerr << "Programma ustoychiva" << endl;
  }
}
```
Bu dastur bajarilganda qo'yidagi ma'lumot ekranga chiqariladi:
S:\> MELTDOWN <ENTER>
Rabotayu! Rabotayu! Rabotayu!
Programma ustoychiva

G'ayri oddiy holat ma'lumot elemenlaridan foydalanish.

Oldingi misollarda dastur catch operatori yordamida qanday g'ayri oddiy holat yuz berganini aniqlashi mumkin edi. G'ayri oddiy holat haqida qancha ko'p ma'lumot olinsa shunchalik hatoga to'g'ri reaktsiya qilishi mumkindir. Misol uchun file_open_error hodisasi yuz berganda hato keltirib chiqargan fayl nomini bilish zarurdir. Shunga uhshab file_read_error va file_write_error dasturda hato joylashgan baytni bilish kerak bo'lishi mumkin. Bunday ma'lumotlarni saklab qolish uchun bu ma'lumotlarni g'ayri oddiy holat sinfiga element sifatida kiritib qo'yish mumkindir. Keyinchalik g'ayri oddiy hodisa yuz berganda dastur bu ma'lumotni qayta ishlovchi funktsiyaga parametr sifatida uzatishshi mumkindir:

```
throw file_open_error(source);
throw file_read_error(344);
```

Gayri oddiy holatni kayta ishlovchida bu parametrlar sinfning mos parametrlariga berib kuyilmishi mumkin. Masalan:

```
class file_open_error
{
public:
  file_open_error(char *filename) { strcpy(file_open_error::filename, filename); }
  char filename[255] ;
};
```

Kutilmagan g'ayri oddiy holatlarni qayta ishlash.

C++ bibliotekalari ma'lum g'ayri oddiy holatlarni qayta ishlovchi funktsiyalarni o'z ichiga oladi. Agar dasturda kuzda tutilmagan g'ayri oddiy hodisa yuz bermasa standart g'ayrioddiy hollarni qayta ishlovchi ishlatiladi. Ko'p hollarda bu standart qayta ishlovchi dastur bajarilishini to'htatib qo'yadi. Qo'yidagi UNCAUGHT.CPP dasturda standart qayta ishlovchining dastur bajarilishini tuhtatishi ko'rsatilgan.:

```
#include <iostream.h>
class some_exception { };
void main(void)
{
  cout << "Pered generatsiey isklyuchitel'noy situatsii" << endl;
  throw some_exception();
  cout << "Isklyuchitel'naya situatsiya sgenerirovana" << endl;
}
```

Bu misolda dastur tomonidan aniqlanmaydigan g'ayri oddiy holat yuz bersa standart qayta ishlovchi chaqiriladi. Shuning uchun ohirgi operator bajarilmaydi. Dasturda mahsus qayta ishlovchidan foydalanish uchun set_unexpected funktsiyasidan foydalanish lozim. Bu funktsiya prototipi except.h sarlavhali faylda aniqlangan.

Funktsiya generatsiya qilgan g’ayri oddiy holatlarni e'lon qilish.

Funktsiya prototipi erdamida shu funktsiya generatsiya qiluvchi g’ayri oddiy holatlarni ko’rsatish mumkin. Agar dastur g’ayri oddiy holatlardan foydalanilsa berilgan funktsiya tomonidan generatsiya qilinuvchi g’ayri oddiy holatlarni ko’rsatish uchun funktsiya prototipidan foydalanish mumkin. Misol uchun qo’yidagi power_plant funktsiyasi prototipi funktsiya melt_down va radiation_leak gayri oddiy holatlarni generatsiya qilishi mumkinligini ko’rsatadi:

```
void power_plant(long power_needed) throw (melt_down, radiation_leak);
```

Bu usul boshka dasturchiga funktsiyadan foydalanilganda qaysi g’ayri oddiy holatlarni tekshirish zarurlishini ko’rsatishga qo’laydir.

G’ayri oddiy holatlar va sinflar.

Sinf yaratganda shu sinfga hos g’ayri oddiy holatlarni ko’rsatish mumkindir. Buning uchun g’ayri oddiy holatni sinfning umumiy (public) elementi sifatida qo’shish lozimdir. Misol uchun qo’yidagi string sinfi ta'rifi ikki g’ayri oddiy holatni aniqlaydi:

```
class string
{
public:
   string(char *str);
   void fill_string(*str);
   void show_string(void);
   int string_length(void);
   class string_empty { } ;
   class string_overflow {};
private:
   int length;
   char string[255];
};
```

Bu sinfda ikki g’ayri oddiy holat string_empty va string_overflow aniqlangan. Dasturda bu holatlar mavjudligini qo’yidagicha tekshirish mumkin:

```
try
{
   some_string.fill_string(some_long_string);
};
catch (string::string_overflow)
{
   cerr << "Previhshena dlina stroki, simvolih otbroshenih" << endl;
}
```