O'ZBEKISTON RESPUBLIKASI OLIY VA
O'RTA MAXSUS TA'LIM VAZIRLIGI

C va C ++ TILI

O'zbekiston Respublikasi Oliy va o'rta maxsus ta'lim vazirligi tomonidan 5521900, 5811200, 5320200, 5523500, 5811300, 5140900 ta'lim yo'nalishlari *talabalari uchun o'quv qo'llanma sifatida tavsiya etilgan*

TOSHKENT
«VORIS-NASHRIYOT»
2013

UO'K 004.43(075)
KBK 32.973.26-018.1
N18

Mualliflar: *Sh.A. Nazirov, R.V. Kabulov,*
M.R. Babajanov, Q.S. Raxmanov.

Taqrizchilar: *A. Haydarov* – O'zbekiston Milliy Universiteti, kafedra mudiri, dotsent;
N.A. Ignatyev – O'zbekiston Milliy Universiteti professori, fizika-matematika fanlari doktori.

Nazirov Sh.A.

C va C++ tili: kasb-hunar kollejlari uchun o'quv qo'llanma / Sh.A. Nazirov, R.V. Kabulov, M.R. Babajanov, Q.S. Raxmanov; O'zbekiston Respublikasi Oliy va o'rta maxsus ta'limi markazi. – Toshkent: «Voris-nashriyot», 2013. – 488 b.

UO'K 004.43(075)
KBK 32.973.26-018.1

Ushbu qo'llanmaning maqsadi – nazariy bilimlarni mustahkamlash, strukturali va obyektga yo'naltirilgan dasturlar yaratish hamda joriy etish amaliy ko'nikmalarini hosil qilishdan iborat.

Qo'llanma dasturlashning asosiy tushunchalarini, ya'ni o'zgaruvchilar, funksiyalar, strukturalar va dinamik xotira bilan ishlash asoslarini yaratishga qaratilgan. Shu bilan birga sinflar, vorislik, amallarni qo'shimcha yuklash, istisnolardan foydalanib obyektli dasturlashga bag'ishlangan.

Qo'llanma o'rta maxsus, kasb-hunar ta'limi hamda oliy o'quv yurtlari professor-o'qituvchilari va talabalari uchun mo'ljallangan.

ISBN 978-9943-4212-0-2

KIRISH

O'quv qo'llanma C va C++ tillarida dasturlashga bag'ishlangan. Ma'lumki, C tili Denni Richchi tomonidan, C++ tili esa B. Straustrup tomonidan yaratilgan.

Ko'p dasturlash tillaridan farqli o'laroq, to 1989-yilgacha C tili standarti mavjud emas edi. Bu davrda ishchi qo'llanma sifatida 1978-yilda bosilib chiqqan B. Kernigan va D. Ritchi kitobidan foydalanilar edi. Odatda, bu kitobga ilovalar K&R maxsus qisqartmasi bilan belgilanadi. Bu kitobning ikkinchi bosmasini ANSI (American National Standards Institute) ishlab chiqqan va ANSI C TuF deb ataluvchi til standartiga moslashtirilgan. Bundan tashqari, ISO S (International Standard Organization S) standarti ham mavjud. Bu standartlar orasida farq ancha kam. Dastur, asosan, ANSIS standartiga asoslangan.

Bern Straustrup (Bjarne Stroustrup), AT&T Bell Laboratories xodimi, 1980-yili C++ tili ustida ish boshladi. Uning hozirgi C tiliga yaqinligini bildiruvchi C++ nomi rasmiy ravishda 1983-yili berildi.

Tilning birinchi tijorat versiyasi 1985-yilning oktabrida taqdim etilib, xuddi shu yili B. Straustrupning «C++ dasturlash tili» (The C++ Programming Language) kitobi bosilib chiqdi.

Til standarti ustida 1990-yili ANSI (American National Standards Institute) ish boshladi. Bu standartning oxirgi varianti 1997-yilning noyabrida e'lon qilindi. Hozirgi davrda C++ tili eng ommaviy dasturlash tiliga aylandi.

O'quv qo'llanmada strukturali dasturlash va obyektli dasturlash usullari chuqur qarab chiqilgan

Qo'llanmada asosiy e'tibor dasturlar tuzish usullariga qaratilgan bo'lib, keltirilgan materiallar ketma-ketlikda berilgan, uning yordamida o'quvchi kompyuterda mustaqil ravishda tez dastur tuzish imkoniga ega bo'ladi

va zamonaviy obyektga yo'naltirilgan dasturlash texnologiyalari bilan tanishadi. Qo'llanma ikki qismdan iborat bo'lib, birinchi qism C tilida strukturali dasturlashga bag'ishlangan. Ikkinchi qism C++ tilida obyektli dasturlashga bag'ishlangan. Har bir qism bir-biridan mustaqil ravishda o'rgatilishi mumkin. Shu maqsadda, ikkinchi qismning birinchi bobida C++ tili asosiy tushunchalari keltirilgan. Qo'llanmani yaratishda yuqorida nomlari keltirilgan til mualliflari kitoblari va bu kitoblarga asoslangan rus hamda chet el mualliflarining keng tarqalgan o'quv qo'llanmalaridan foydalaniladi.

O'ylaymizki, qo'llanma bilan tanishgan o'rta maxsus, kasb-hunar ta'limi o'quvchilari, oliy o'quv yurtlari talabalari, magistrlari va aspirantlari kompyuterda C va C++ tilida o'z dasturlarini yaratishga kirishadilar.

I QISM. C TILIDA STRUKTURALI DASTURLASH

1-bob. TIL LEKSIK ASOSLARI

1.1. Alifbo va xizmatchi so'zlar

Alifbo. C tili alifbosiga quyidagi simvollar kiradi.

- Lotin alifbosining katta va kichik harflari (A, B, ..., Z, a, b, ..., z).
- Raqamlar: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Maxsus simvollar: ', {} | [] () + – / % \; ' : ? <=> _ ! & * # ~ ^
- Ko'rinmaydigan simvollar ("umumlashgan bo'shliq simvollari»). Leksemalarni o'zaro ajratish uchun ishlatiladigan simvollar (misol uchun bo'shliq, tabulatsiya, yangi qatorga o'tish belgilari).

Izohlarda, satrlarda va simvolli konstantalarda boshqa literallar, masalan, rus harflari ishlatilishi mumkin.

C tilida oltì xil turdagi leksemalar ishlatiladi: erkin tanlanadigan va ishlatiladigan identifikatorlar, xizmatchi so'zlar, konstantalar (konstanta satrlar), amallar (amallar belgilari) va ajratuvchi belgilar.

Identifikator. Identifikatorlar lotin harflari, ostki chiziq belgisi va sonlar ketma-ketligidan iborat bo'ladi. Identifikator lotin harfidan yoki ostki chiziq belgisidan boshlanishi lozim.

Misol uchun:

A1, _MAX, adress_01, RIM, rim

Katta va kichik harflar farqlanadi, shuning uchun oxirgi ikki identifikator bir-biridan farq qiladi.

Borland kompilatorlaridan foydalanilganda nomning birinchi 32 harfi, ba'zi kompilatorlarda 8 ta harfni inobatga oladi. Bu holda NUMBER_OF_TEST va NUMBER_OF_ROOM identifikatorlari bir-biridan farq qilmaydi.

Xizmatchi so'zlar. Tilda ishlatiluvchi, ya'ni dasturchi tomonidan o'zgaruvchilar nomlari sifatida ishlatish mumkin bo'lmagan identifikatorlar xizmatchi so'zlar deyiladi.

C tilida quyidagi xizmatchi so'zlar mavjud:

Turlar nomlari: char, short, int, unsigned, long, float, double.

Operatorlar nomlari: if, else, switch, case, while, do, for, default, break, continue, goto.

Xotira turlari: auto, register, static, extern.

Turlar bilan ishlash: typedef, sizeof.

Struktura: struct,union.

Chiqish: return, entry.

1.2. O'zgaruvchilar

O'zgaruvchilar obyekt sifatida. C tilining asosiy tushunchalaridan biri nomlangan xotira qismi – obyekt tushunchasidir. Obyektning xususiy holi bu o'zgaruvchidir. O'zgaruvchiga qiymat berilganda unga ajratilgan xotira qismiga shu qiymat kodi yoziladi. O'zgaruvchi qiymatiga nomi orqali murojaat qilish mumkin, xotira qismiga esa faqat adresi orqali murojaat qilinadi. O'zgaruvchi nomi, bu – erkin kiritiladigan identifikatordir. O'zgaruvchi nomi sifatida xizmatchi so'zlarni ishlatish mumkin emas.

O'zgaruvchilarni ta'riflash. C tilida o'zgaruvchini aniqlash uchun kompyuterga uning turi (masalan, int, char yoki float) hamda ismi haqida ma'lumot beriladi. Bu axborot asosida kompilatorga o'zgaruvchi uchun qancha joy ajratish lozim va bu o'zgaruvchida qanday turdagi qiymat saqlanishi mumkinligi haqida ma'lumot aniq bo'ladi. O'zgaruvchi nomi identifikator bo'lib, xizmatchi so'zlardan farqli bo'lishi kerak.

Har bir yacheyka bir bayt o'lchovga ega. Agar o'zgaruvchi uchun ko'rsatilgan tur 4 baytni talab qilsa, uning uchun to'rtta yacheyka ajratiladi. Aynan o'zgaruvchining turiga muvofiq ravishda kompilator bu o'zgaruvchi uchun qancha joy ajratish kerakligini aniqlaydi.

Kompyuterda qiymatlarni ifodalash uchun bitlar va baytlar qo'llaniladi va xotira baytlarda hisoblanadi.

O'zgaruvchilarning turlari. O'zgaruvchilarning quyidagi turlari mavjud:

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 turlar oldiga unsigned (ishorasiz) ta'rifi qo'shilishi mumkin. Bu ta'rif qo'shilgan butun sonlar ustida amallar mod 2^n arifmetikasiga asoslangan. Bu yerda n soni int turi xotirada egallovchi razryadlar sonidir. Agar ishorasiz k soni uzunligi int soni razryadlar sonidan uzun bo'lsa, bu son qiymati k mod 2^n ga teng bo'ladi. Ishorasiz k son uchun ga –k amali 2^n–k formula asosida hisoblanadi. Ishorali, ya'ni signed turidagi sonlarning eng katta razryadi son ishorasini ko'rsatish uchun ishlatilsa, unsigned (ishorasiz) turdagi sonlarda bu razryad sonni tasvirlash uchun ishlatiladi.

O'zgaruvchilarni dasturning ixtiyoriy qismida ta'riflash yoki qayta ta'riflash mumkin.

Misol uchun:

int a, bl, ac; yoki

int a;

int bl;

int ac;

O'zgaruvchilar ta'riflanganda ularning qiymatlari aniqlanmagan bo'ladi. Lekin o'zgaruvchilarni ta'riflashda initsializatsiya, ya'ni boshlang'ich qiymatlarini ko'rsatish mumkin.

Misol uchun:

int i=0;

char c='k';

Typedef ta'riflovchisi yangi turlarni kiritishga imkon beradi.

Misol uchun yangi COD turini kiritish:

typedef unsigned char COD;

COD simbol;

Butun turlar o'lchami. Bir xil turdagi o'zgaruvchilar uchun turli kompyuterlarda xotiradan turli hajmdagi joy ajratilishi mumkin. Lekin bitta kompyuterda bir xil turdagi ikkita o'zgaruvchi bir xil miqdorda joy egallaydi.

Masalan, char turli o'zgaruvchi bir bayt hajmni egallaydi. Ko'pgina kompyuterlarda short int (qisqa butun) turi ikki bayt, long int turi esa 4 bayt joy egallaydi. Butun qiymatlar o'lchovini kompyuter sistemasi va ishlatiladigan kompilator aniqlaydi. 32 razryadli kompyuterlarda butun o'zgaruvchilar 4 bayt joy egallaydi.

1.3. Konstantalar

Konstantalar turlari. Konstanta, bu – o'zgartirish mumkin bo'lmagan qiymatdir. C tilida besh turdagi konstantalar ishlatilishi mumkin: simvollar, butun sonlar, haqiqiy sonlar, sanovchi konstantalar va nol ko'rsatkich.

Belgili o'zgarmaslar. Belgili o'zgarmaslar, odatda, bir bayt joyni egallaydi va bu 256 xil belgini saqlash uchun yetarlidir. Char turi qiymatlarini 0...255 sonlar to'plamiga yoki ASCII belgilar to'plamiga interpretatsiya qilish mumkin.

ASCII belgilari deganda kompyuterlarda qo'llaniladigan standart belgilar to'plami tushuniladi. ASCII – bu American Standard Code for Information Interchange (Amerikaning axborot almashinishi uchun standart kodi) degan ma'noni anglatadi.

Misol uchun: 'x','*','\012','\0','\n' – bitta simvolli konstanta; 'dd','\n\t','\x07\x07' – ikki simvolli konstantalar.

C kompilatorida tekstlarni formatlovchi bir nechta maxsus belgilardan foydalaniladi. (Ulardan eng ko'p tarqalgani jadvalda keltirilgan.)

Maxsus belgilar axborotlarni ekranga, faylga va boshqa chiqarish qurilmalariga chiqarishda formatlash uchun qo'llaniladi.

Maxsus '\' simvolidan boshlangan simvollar eskeyp simvollar deyiladi. 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 (bacspase)	Bir qadam qaytish
\f	0x0C	ff (form feed)	Sahifani o'tkazish
\n	0x0A	lf (line feed)	Qatorni o'tkazish
\r	0x0D	cr (carriage return)	Karetkani qaytarish
\t	0x09	ht (horizontal tab)	Gorizontal tabulatsiya
\v	0x0B	vt (vertical tab)	Vertikal tabulatsiya

\\	0x5C	\ (bacslash)	Teskari chiziq
\'	0x27	' (single out)	Apostrif (oddiy qavs)
\"	0x22	«(double quote)	Ikkilik qavs
\?	0x3F	? (question mark)	Savol belgisi
\000	000	ixtiyoriy (octal number)	Simvol sakkizlik kodi
\xhh	0xhh	ixtiyoriy (hex number)	Simvol o'n oltilik kodi

Ma'lumotlarning butun son turi. Butun sonlar o'nlik, sakkizlik yoki o'n oltilik sanoq sistemalarida berilishi mumkin.

O'nlik sanoq sistemasida butun sonlar 0–9 raqamlari ketma-ketligidan iborat bo'lib, birinchi raqami 0 bo'lishi kerak emas. Lekin yagona 0 bo'lishi mumkin.

Sakkizlik sanoq sistemasida butun sonlar 0 bilan boshlanuvchi 0–7 raqamlaridan iborat ketma-ketlikdir.

O'n oltilik sanoq sistemasida butun son 0x yoki 0X bilan boshlanuvchi 0–9 raqamlari va a–f yoki A–F harflaridan iborat ketma-ketlikdir.

Masalan, 15 va 22 o'nlik sonlari sakkizlikda 017 va 026, o'n oltilikda 0xF va 0x16 shaklda tasvirlanadi.

Ma'lumotlarning uzun butun son turi:

Oxiriga l yoki L harflari qo'yilgan o'nlik, sakkizlik yoki o'n oltilik butun son.

Ma'lumotlarning ishorasiz (unsigned) butun son turi:

Oxiriga u yoki U harflari qo'yilgan o'nlik, sakkizlik yoki o'n oltilik oddiy yoki uzun butun son.

Ma'lumotlarning haqiqiy son turi. Ma'lumotlarning haqiqiy son turi olti qismdan iborat bo'lishi mumkin: butun qism, nuqta, kasr qism, e yoki E belgisi, o'nlik daraja va F yoki f suffikslari.

Masalan: 66., .0, .12, 3.14F, 1.12e-12.

Ma'lumotlarning uzun haqiqiy son turi:

Oxiriga L yoki l suffikslari qo'yilgan haqiqiy son.

Masalan: 2E+6L;

Sanovchi konstanta. Sanovchi konstantalar enum xizmatchi so'zi yordamida kiritilib, int turidagi sonlarga qulay so'zlarni mos qo'yish uchun ishlatiladi.

Misol uchun:

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

Agar son qiymatlari ko'rsatilmagan bo'lsa, eng chapki so'zga 0 qiymati berilib, qolganlariga tartib bo'yicha o'suvchi sonlar mos qo'yiladi:

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 qiymatlarni 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;

Nol ko'rsatkich. NULL ko'rsatkich yagona arifmetik bo'lmagan konstantadir. Konkret realizatsiyalarda noll ko'rsatkich 0 yoki 0L, yoki nomlangan konstanta NULL orqali tasvirlanishi mumkin. Shuni aytish lozimki, bu konstanta qiymati 0 bo'lishi yoki '0' simvoli kodiga mos kelishi shart emas.

Mantiqiy konstanta. Mantiqiy konstantalar true (rost) va false (yolg'on) qiymatlardan iborat. C tilida butun sonlar va ifodalar mantiqiy konstantalar sifatida qaraladi. Ichki ko'rinishi false – 0, ixtiyoriy boshqa qiymat true deb qaraladi.

Satrli konstanta. Satrli konstantalar C tili konstantalariga kirmaydi, balki leksemalari alohida turi hisoblanadi. Shuning uchun adabiyotlarda satrli konstantalar satrli leksemalar deb ham ataladi.

Satrli konstanta, bu – ikkilik qavslarga olingan ixtiyoriy simvollar ketma-ketligidir. Misol uchun "Men satrli konstantaman".

Satrlar orasiga eskeyp simvollar ham kirishi mumkin. Bu simvollar oldiga \ belgisi qo'yiladi. Misol uchun:

"\n Bu satr \n uch qatorga \n joylashadi".

Satr simvollari xotirada ketma-ket joylashtiriladi va har bir satrli konstanta oxiriga avtomatik ravishda kompilator tomonidan '\0' simvoli

qo'shiladi. Shunday satrning xotiradagi hajmi simvollar soni +1 baytga tengdir.

Ketma-ket kelgan va bo'shliq, tabulatsiya yoki satr oxiri belgisi bilan ajratilgan satrlar kompilatsiya davrida bitta satrga aylantiriladi. Misol uchun:

"Salom" "Toshkent"

satrlari bitta satr deb qaraladi.

"Salom Toshkent"

Bu qoidaga bir necha qatorga yozilgan satrlar ham bo'ysunadi. 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 oxiri belgisigacha bo'shliq belgisi kelsa, bu bo'shliq belgilari '\' va '\n' belgisi bilan birga satrdan o'chiriladi. Satrning o'zi keyingi satrda kelgan satr bilan qo'shiladi.

"O'zbekistonga \

bahor \

keldi"

qatorlari bitta qatorga mos:

"O'zbekistonga bahor keldi"

Nomlangan konstantalar. C tilida o'zgaruvchilardan tashqari nomlangan konstantalar kiritilishi mumkin. Bu konstantalar qiymatlarini dasturda o'zgartirish mumkin emas. Konstamalar nomlari dasturchi tomonidan kiritilgan va xizmatchi so'zlardan farqli bo'lgan identifikatorlar bo'lishi mumkin. Odatda, nom sifatida katta lotin harflari va ostiga chizish belgilari kombinatsiyasidan iborat identifikatorlar ishlatiladi. Nomlangan konstantalar quyidagi shaklda kiritiladi:

const tur konstanta_nomi=konstanta_qiymati.

Misol uchun:

const double EULER=2.718282;

const long M=99999999;

const R=765;

Oxirgi misolda konstanta turi ko'rsatilmagan, bu konstanta int turiga tegishli deb hisoblanadi.

1.4. Amallar

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'shish va – ayirish amallariga hamda multiplikativ, ya'ni * ko'paytirish, / bo'lish va % modul olish amallariga ajratiladi.

Butun sonni butun songa bo'lganda natija butun songacha yaxlitlanadi. Misol uchun, 20/3=6; (–20)/3=–6; 20/(–3)=–6.

Modul amali butun sonni butun songa bo'lishdan hosil bo'ladigan qoldiqqa tengdir. Agar modul amali musbat operandlarga qo'llanilsa, natija ham musbat bo'ladi, aks holda natija ishorasi kompilatorga bog'liqdir.

Unar amallarga ishorani o'zgartiruvchi unar minus – va unar plus + amallari kiradi. Bundan tashqari inkrement ++ va dekrement – – amallari ham unar amallarga kiradi.

Inkrement ++ 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 esa i++ ko'rinishda ishlatish oldin o'zgaruvchi qiymatidan foydalanib, so'ngra oshirish kerakligini ko'rsatadi. Misol uchun, i ning 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 ning qiymati 3 ga teng bo'ladi.

Dekrement – – 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 mumkin.

Amallar ustivorligi. Murakkab ifodalarda qaysi amal birinchi navbatda bajarilishi operator prioritetiga bog'liq.

Masalan: $x=5+3*8$.

Ko'paytirish qo'shishga nisbatan yuqoriroq prioritetga ega. Shuning uchun bu ifoda qiymati 29 ga teng bo'ladi.

Agarda ikkita matematik ifodaning prioriteti teng bo'lsa, ular chapdan o'ngga qarab ketma-ket bajariladi.

Masalan: $x=5+3+8*9+6*4$.

Bu ifodada birinchi ko'paytirish amallari chapdan o'ngga qarab bajariladi: 8*9=72 va 6*4=24. Keyin qo'shish amallari bajariladi. Natija $x=104$ qiymatga ega bo'ladi.

Lekin barcha amallar ham bu tartibga amal qilmaydi. Masalan, o'zlashtirish amali o'ngdan chapga qarab bajariladi.

Additiv amallarning ustivorligi multiplikativ amallarning ustivorligidan pastroqdir.

Unar amallarning ustivorligi binar amallardan yuqoriroqdir.

Razryadli amallar. Razryadli amallar natijasi butun sonlar ikkilik ko'rinishlarining har bir razryadiga mos mantiqiy amallarni qo'llashdan hosil bo'ladi. Masalan, 5 kodi 101 ga teng va 6 kodi 110 ga teng.

6&5 qiymati 4 ga, ya'ni 100 ga teng.

6|5 qiymati 7 ga, ya'ni 111 ga teng.

6^5 qiymati 3 ga, ya'ni 011 ga teng.

Bu misollarda amallar ustivorligi oshib borishi tartibida berilgan.

Bu amallardan tashqari M<<N chapga razryadli siljitish va M>>N o'ngga 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-darajasiga ko'paytirishga mos keladi. Misol uchun 5<<2=20. Bu amalning bitli ko'rinishi: 101<<2=10100.

Agar operand musbat bo'lsa, N pozitsiyaga o'ngga surish chap operandni ikkining N-darajasiga bo'lib, kasr qismini tashlab yuborishga mosdir. Misol uchun: 5>>2=1. Bu amalning bitli ko'rinishi: 101>>2=001=1. Agarda operand qiymati manfiy bo'lsa, ikki variant mavjuddir: arifmetik siljitishda bo'shatilayotgan razryadlar ishora razryadi qiymati bilan to'ldiriladi, mantiqiy siljitishda bo'shatilayotgan razryadlar nollar bilan to'ldiriladi.

Razryadli surish amallarining ustivorligi o'zaro teng, razryadli inkor amalidan past, qolgan razryadli amallardan yuqoridir. Razryadli inkor amali unar amalga, qolgan amallar binar amallarga kiradi.

Nisbat amallari. Nisbat amallari qiymatlari 1 ga teng, agar nisbat bajarilsa va aksincha, 0 ga tengdir. Nisbat amallari arifmetik turdagi operandlarga yoki ko'rsatkichlarga qo'llaniladi.

Misollar:

!!=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 yoki teng >=, kichik yoki teng <= amallarining ustivorligi bir xildir.

Teng== va teng emas !=amallarining ustivorligi o'zaro teng va qolgan amallardan pastdir.

Mantiqiy amallar. C tilida mantiqiy tur yo'q. Shuning uchun mantiqiy amallar butun sonlarga qo'llaniladi. Bu amallarning natijalari quyidagicha aniqlanadi:

x||y amali 1 ga teng, agar x>0 yoki 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 bo'lsa, aksincha, 0 ga teng.

Bu misollarda amallar ustivorligi oshib borish tartibida berilgan.

Inkor! amali unar, qolganlari binar amallardir.

Qiymat berish amali. Qiymat berish amali=binar amal bo'lib, chap operandi, odatda, o'zgaruvchi, o'ng operandi esa 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 oxiriga nuqtali vergul (;) belgisi qo'yilganda 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 tilida murakkab qiymat berish amali mavjud bo'lib, umumiy ko'rinishi quyidagicha:

O'zgaruvchi_nomi **amal**=ifoda;

Bu yerda **amal** quyidagi amallardan biri *, /, %, +, –, &, ^, |, <<, >>.

Misol uchun:

x+=4 ifoda x=x+4 ifodaga ekvivalent;

x*=a ifoda x=x* a ifodaga ekvivalent;

x/=a+b ifoda x=x/(a+b) ifodaga ekvivalent;

x>>=4 ifoda x=x>>4 ifodaga ekvivalent;

Imlo belgilari amal sifatida. C tilida ba'zi bir imlo belgilari ham amal sifatida ishlatilishi mumkin. Bu belgilar oddiy ( ) va kvadrat [ ] qavslardir. Oddiy qavslar binar amal deb qaralib, ifodalarga yoki funksiyaga murojaat qilishda foydalaniladi. Funksiyaga murojaat qilish quyidagi shaklda amalga oshiriladi:

<funksiya nomi> (<argumentlar ro'yxati>). Misol uchun sin(x) yoki max(a, b).

Kvadrat qavslardan massivlarga murojaat qilishda foydalaniladi. Bu murojaat quyidagicha amalga oshiriladi:

<massiv nomi>[<indeks>]. Misol uchun, a[5] yoki b[n][m].

Vergul simvolini ajratuvchi belgi sifatida ham, amal sifatida ham qarash mumkin. Vergul bilan ajratilgan amallar ketma-ketligi bir amal deb qaralib, chapdan o'ngga hisoblanadi va oxirgi ifoda qiymati natija deb qaraladi. Misol uchun:

d=4, d+2 amali natijasi 6 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 turga, a N – INT turga tegishli bo'lsa, (N>0) ? F : N ifoda N musbat yoki manfiyligidan qat'iy nazar, DOUBLE turiga tegishli bo'ladi.

Shartli ifodada birinchi ifodani qavsga olish shart emas.

Amallar ustivorligi jadvali

Rang	Amallar	Yo'nalish
1	() [] -> :: .	Chapdan o'ngga
2	! ~ + - ++ -- & * (tur) sizeof new delete tur()	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)	O'ngdan chapga
15	= *= /= %= += -= &= ^= |= <<= >>=	O'ngdan chapga
16	, (vergul amali)	Chapdan o'ngga

1.5. Turlar bilan ishlash

Turlarni keltirish. Turlarni keltirish (type casting) ma'lum turdagi o'zgaruvchi boshqa turdagi qiymat qabul qilganda foydalaniladi. Ba'zi turlar uchun keltirish avtomatik ravishda bajariladi. Avtomatik turlarni keltirish o'zgaruvchi turining hajmi qiymatni saqlashga yetarli bo'lganda bajariladi. Bu jarayon kengaytirish *(widening)* yoki yuksaltirish *(promotion)* deb ataladi, chunki kichik razryadli tur katta razryadli turga kengaytiriladi. Bu holda turlarni avtomatik keltirish xavfsiz deb ataladi. Masalan, int turi char turidagi qiymatni saqlashga yetarli, shuning uchun turlarni keltirish talab qilinmaydi. Teskari jarayon toraytirish *(narrowing)* deb ataladi, chunki qiymatni o'zgartirish talab etiladi. Bu holda turlarni avtomatik keltirish xavfli deb ataladi. Masalan, haqiqiy turni butun turga keltirilganda kasr qism tashlab yuboriladi.

Amallarda turlarni avtomatik keltirish. Binar arifmetik amallar bajarilganda turlarni keltirish quyidagi qoidalar asosida amalga oshiriladi:

Short va **char** turlari **int** turiga keltiriladi;

agar operandlardan biri **long** turiga tegishli bo'lsa, ikkinchi operand ham long turiga keltiriladi va natija ham **long** turiga tegishli bo'ladi;

agar operandlardan biri **float** turiga tegishli bo'lsa, ikkinchi operand ham **float** turiga keltiriladi va natija ham **float** turiga tegishli bo'ladi;

agar operandlardan biri **double** turiga tegishli bo'lsa, ikkinchi operand ham **double** turiga keltiriladi va natija ham **double** turiga tegishli bo'ladi;

agar operandlardan biri **long double** turiga tegishli bo'lsa, ikkinchi operand ham **long double** turiga keltiriladi va natija ham **long double** turiga tegishli bo'ladi.

Ifodalarda turlarni avtomatik keltirish. Agar ifodada short va int turidagi o'zgaruvchilar ishlatilsa, butun ifoda turi int ga ko'tariladi. Agar ifodada biror o'zgaruvchi turi long bo'lsa, butun ifoda turi long turga ko'tariladi. Ko'zda tutilganidek, hamma butun konstantalar int turiga ega deb qaraladi. Hamma butun konstantalar oxirida L yoki l simvoli turgan bo'lsa, long turiga ega.

Agar ifoda float turidagi operandga ega bo'lsa, butun ifoda float turiga ko'tariladi. Agar biror operand double turiga ega bo'lsa, butun ifoda turi double turiga ko'tariladi.

Turlar bilan ishlovchi amallar. Turlarni o'zgartirish amali quyidagi ko'rinishga ega:

(tur_nomi) operand;

Bu amal operandlar qiymatini ko'rsatilgan turga keltirish uchun ishlatiladi. Operand sifatida konstanta, o'zgaruvchi yoki qavslarga olingan ifoda kelishi mumkin. Misol uchun (long)6 amali konstanta qiymatini o'zgartirmagan holda operativ xotirada egallagan baytlar sonini oshiradi. Bu misolda konstanta turi o'zgarmagan bo'lsa, (double)6 yoki (float)6 amali konstanta ichki ko'rinishini ham o'zgartiradi. Katta butun sonlar haqiqiy turga keltirilganda sonning aniqligi yo'qolishi mumkin.

Masalan:

```
int x=1.7+1.8;
int y=(int)1.7+(int)1.8;
```

Bu amallar bajarilishi natijasida x o'zgaruvchi qiymati 3 ga, y o'zgaruvchi qiymati ikkiga teng bo'ladi.

sizeof amali operand sifatida ko'rsatilgan obyektning baytlarda xotiradagi hajmini hisoblash uchun ishlatiladi. Bu amalning ikki ko'rinishi mavjud:

sizeof ifoda

sizeof (tur)

Shuni ta'kidlab o'tish lozimki, sizeof funksiyasi preprotsessor qayta ishlash jarayonida bajariladi, shuning uchun dastur bajarilish jarayonida vaqt talab etmaydi.

Misol uchun:

sizeof (3.14)=8

sizeof (3.14f)=4

sizeof (3.14L)=10

sizeof(char)=1

sizeof(double)=8.

1.6. C tilida dastur tuzilishi

Sodda dastur tuzilishi. Dastur preprotsessor komandalari va bir necha funksiyalardan iborat bo'lishi mumkin. Bu funksiyalar orasida main nomli asosiy funksiya bo'lishi shart. Agar asosiy funksiyadan boshqa funksiyalar ishlatilmasa, dastur quyidagi ko'rinishda tuziladi:

Preprosessor_komandalari

void main()

{

Dastur tanasi.

}

Preprosessor direktivalari kompilatsiya jarayonidan oldin preprotsessor tomonidan bajariladi. Natijada dastur matni preprotsessor direktivalari asosida o'zgartiriladi.

Preprosessor komandalaridan ikkitasini ko'rib chiqamiz.

#include <fayl_nomi> Bu direktiva standart kutubxonalardagi funksiyalarni dasturga joylash 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 <stdio.h>

void main()

{

printf("n Salom, Dunyo! \n");

}

Bu dastur ekranga **Salom, Dunyo!** jumlasini chiqaradi.

Almashtiruvchi define direktivasi yordamida bu dasturni quyidagicha yozish mumkin:

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

Almashtiruvchi define direktivasidan nomlangan konstantalar kiritish uchun foydalanish mumkindir.

Misol uchun:

#define ZERO 0

Agar dasturda quyidagi matn mavjud bo'lsa:

int d=ZERO;

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

int d=0;

1.7. Ma'lumotlarni kiritish va chiqarish

Formatli chiqarish – printf. Chiqarish printf funksiyasi ko'rsatilgan parametrlarni standart oqimga chiqarish uchun ishlatiladi. Standart oqim tushunchasi keyingi boblarda yoritiladi. Hozircha standart oqim sifatida monitor tushunilishi yetarli.

Funksiya **stdio.h** modulida joylashgan bo'lib, umumiy ko'rinishi quyidagicha:

printf (control, arg1, arg2, ...)

Bunda 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.

O'zgartirish simvollari quyidagilardan iborat.

Butun sonlar uchun:

d – parametr ishorali o'nlik butun songa aylantiriladi.

u – parametr ishorasiz o'nlik butun songa aylantiriladi.

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

x – parametr ishorasiz va 0x belgisiz o'n oltilik songa aylantiriladi.

X – parametr xuddi x kabi. Faqat harf bilan ko'rsatiluvchi raqamlar katta harf, ya'ni A, B, C, D, E, F sifatida yoziladi.

Haqiqiy sonlar uchun:

e – parametr float yoki double turidagi son deb qaraladi va ishorali m.nnnnnne+-xx ko'rinishidagi o'nlik songa keltiriladi.

E – parametr xuddi e kabi. Faqat mantissa belgisi katta harf, ya'ni E sifatida yoziladi.

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

g – parametr berilgan son qiymati va aniqligi uchun eng ixcham %e yoki %f tanlaydi.

G – parametr xuddi g kabi. Faqat mantissa belgisi katta harf, ya'ni E sifatida yoziladi.

Simvol va satr uchun:

c – parametr bitta simvol deb qaraladi.

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

Misol:

```
#include <stdio.h>
int main()
{
int num=-27; int number=27; float f=123.456;
char r='a'; char str[4]=«abc»;
printf("%d\n",    num);     /*  -27          */
printf("%u\n",    number);  /*  27           */
printf("%o\n",    number);  /*  33           */
printf("%x\n",    number);  /*  lb           */
printf("%f\n",    f);       /*  123.456001   */
```



```
printf("%e\n",    f);       /*  1.23456e+02  */
printf("%E\n",    f);       /*  1.23456E+02  */
printf("%c\n",    r);       /*  a            */
printf("%s\n", str);        /*  abc          */
return 0;
}
```

Protsent % belgisi va o'zgartirish simvoli orasiga quyidagi 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 turidagi sonlarda nuqtadan keyin qancha kasr raqamlari bosib chiqarilishini ko'rsatuvchi raqamlar ketma-ketligi.

Chiqarilayotgan son long turiga tegishli ekanligini ko'rsatuvchi uzun o'nlik markeri l.

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

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

Quyidagi jadval har xil spetsifikatsiyalarni «HELLO, WORLD» (12 simvolli) so'zini bosishga ta'sirini ko'rsatadi. Bu yerda har bir maydon uzunligini ko'rsatish uchun maydon oxiriga 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:
```

Turlar maksimal va minimal qiymatlari. Turli turlar maksimal va minimal qiymatlari <LIMITS.H> faylidagi konstantalarda saqlanadi.

Quyidagi dasturda char turidagi bitlar soni va char turi maksimal va minimal qiymati ekranga chiqariladi:

```
#include<stdio.h>
#include<limits.h>
```

```
int main()
{
printf("CHAR_BIT=%d\n", CHAR_BIT);
printf("CHAR_MIN=%d CHAR_MAX=%d\n", CHAR_MIN,
CHAR_MAX);
return 0;
}
```

Quyidagi dasturda short, int, long turlari maksimal va minimal qiymati ekranga chiqariladi:

```
#include<stdio.h>
#include<limits.h>
int main()
{
printf("SHRT_MIN=%d SHRT_MAX=%d\n", SHRT_MIN,
SHRT_MAX);
printf("INT_MIN=%d INT_MAX=%d\n", INT_MIN, INT_
MAX);
printf("LONG_MIN=%ld LONG_MAX=%ld\n", LONG_MIN,
LONG_MAX);
return 0;
}
```

Quyidagi dasturda unsigned char, unsigned short, unsigned int, unsigned long turlari maksimal va minimal qiymati ekranga chiqariladi:

```
#include<stdio.h>
#include<limits.h>
int main()
{
printf("UCHAR_MAX=%u\n", UCHAR_MAX);
printf("USHRT_MAX=%u\n", USHRT_MAX);
printf("UINT_MAX=%u\n", INT_MAX);
printf("ULONG_MAX=%ul\n", ULONG_MAX);
return 0;
}
```

Formatli kiritish scanf. scanf funksiyasi stdio.h modulida joylashgan bo'lib, umumiy ko'rinishi quyidagicha:

scanf(control, arg1, arg2, ...)

Funksiya standart oqimdan simvollarni o'qib, boshqaruvchi qator asosida formatlab, mos parametrlarga yozib qo'yadi. Parametr ko'rsatkich bo'lishi lozim.

Boshqaruvchi qator quyidagi o'zgartirish spetsifikatsiyalaridan iborat:

bo'shliq, tabulatsiya, keyingi qatorga o'tish simvollari;

oddiy simvollar (% dan tashqari) 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.

Quyidagi spetsifikatsiya simvollarini ishlatish mumkin:

d – ishorali o'nli butun son kutilmoqda;

o – ishorali sakkizlik butun son kutilmoqda;

x – ishorali o'n oltilik butun son kutilmoqda;

h – ishorasiz o'nlik son kutilmoqda;

c – bitta simvol kutilmoqda;

s – satr kutilmoqda;

f – float turidagi son kutilmoqda.

Kiritilayotgan sonning butun raqamlari va nuqtadan so'ng kasr raqamlari soni va E yoki e belgisidan so'ng mantissa raqamlari soni ko'rsatilishi mumkin.

```
#include <stdio.h>
int main(void)
{
unsigned width, precision;
int number=256;
double weight=242.5;
printf("What field width?\n");
scanf("%d", &width);
printf("The number is:%*d:\n", width, number);
printf("Now enter a width and a precision:\n");
scanf("%d %d", &width, &precision);
printf("Weight=%*.*f\n", width, precision, weight);
printf("Done!\n");
```

```
return 0;
}
```

Lokal va global o'zgaruvchilar. C tilida o'zgaruvchi ta'rifi albatta blok boshida joylashishi lozim.

O'zgaruvchi mavjudlik sohasi deb shu o'zgaruvchiga ajratilgan xotira mavjud bo'lgan dastur qismiga aytiladi. O'zgaruvchi ko'rinish sohasi deb o'zgaruvchi qiymatini olish mumkin bo'lgan dastur qismiga aytiladi. Biror blokda ta'riflangan o'zgaruvchi lokal o'zgaruvchi deyiladi. Har qanday blokdan tashqarida ta'riflangan o'zgaruvchi global o'zgaruvchi deyiladi.

Lokal o'zgaruvchi mavjudlik va ko'rinish sohasi ta'rifdan, to shu ta'rif joylashgan blok oxirigachadir.

Tashqi blokdagi o'zgaruvchi nomi shu blokda joylashgan yoki shu blokdagi ichki blokda o'zgaruvchi nomi bilan bir xil bo'lmasligi kerak.

Global o'zgaruvchi mavjudlik sohasi ta'rifdan, to dastur oxirigacha.

Agar ichki blokdagi o'zgaruvchi nomi global o'zgaruvchi nomi bilan bir xil bo'lsa, lokal o'zgaruvchi ko'rinish sohasida global o'zgaruvchi ko'rinmay qoladi.

Misol:

```
#include<stdio.h>
int i=5;
int k=6;
int main()
{
int i=9;
printf("%d\n", i);
printf("%d\n", k);
return 0;
}
```

Natija:

```
9
6
```

1-bob bo'yicha savollar

1. Butun va haqiqiy turlarning qanday farqi bor?
2. Ishorasiz unsigned turining xossalarini ko'rsating.

3. Ishorasiz unsigned short int va long int turlarining o'zaro farqi nimada?
4. Birinchi qaysi funksiya bajariladi?
5. Simvolli kiritish funksiyalari.
6. Shartli amalning umumiy ko'rinishi.
7. Turlarni keltirish qoidalari.
8. Quyidagi #include direktivasi qanday vazifani bajaradi?
9. Bosh main() funksiyasining o'ziga xos xususiyati nimadan iborat?
10. Izohlar bir necha qatorda yozilishi mumkinmi?

1-bob bo'yicha topshiriqlar

1. Matematik amallardan foydalanishni ko'rsatuvchi dastur tuzing.
2. Mantiqiy amallardan foydalanishni ko'rsatuvchi dastur tuzing.
3. Nisbat amallardan foydalanishni ko'rsatuvchi dastur tuzing.
4. Munosabat amallaridan foydalanishni ko'rsatuvchi dastur tuzing.
5. Sonning absolut qiymatini shartli amal yordamida hisoblovchi dastur tuzing.

2-bob. OPERATORLAR VA FUNKSIYALAR

2.1. Operatorlar turlari

Operatorlar va bloklar. Har qanday dastur funksiyalar ketmaketligidan iborat bo'ladi. Funksiyalar sarlavha va funksiya tanasidan iborat bo'ladi. Funksiya sarlavhasiga void main() ifoda misol bo'la oladi. Funksiya tanasi obyektlar ta'riflari va operatorlardan iborat bo'ladi.

Har qanday operator nuqtali-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 ixtiyoriy jumla yozish mumkin. Kompilator bu jumlani tekshirib o'tirmaydi. Izoh operatoridan dasturni tushunarli qilish maqsadida izohlar kiritish uchun foydalaniladi.

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

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

tanlash operatorlari;
sikl operatorlari;
o'tish operatorlari.

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

```
i++;
x*=i;
i=x-4*i;
```


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

Quyidagi dastur qismi qo'shma operator:

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

Bu fragment esa blok:

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

2.2. Tanlash operatorlari

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

```
if (ifoda)
1-operator
else
2-operator
yoki
if (ifoda)
1-operator
```

Shartli operator bajarilganda avval ifoda hisoblanadi. Agar qiymat rost, ya'ni noldan farqli bo'lsa, 1-operator bajariladi. Agar qiymat yolg'on, ya'ni nol bo'lsa va else ishlatilsa, 2-operator bajariladi. Operatorning else qismi 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 qo'yish 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 dasturi:

```
#include<stdio.h>
int main()
{
float a, b, c, max;
scanf("%f", &a);
scanf("%f", &b);
scanf("%f", &c);
if (a>b)
if (a>c) max=a; else max=c;
else
if (b>c) max=b; else max=c;
printf("\n max=%f", max);
return 0;
}
```

Keyingi misolda kiritilgan ball va maksimal ball asosida baho aniqlanadi:

```
#include<stdio.h>
int main()
{
int ball, max_ball, baho;
printf("\n ball=");
scanf("%d", &ball);
printf("\n max_ball=");
```



```
scanf("%d", &max_ball);
float d=(float)ball/max_ball;
if (d>0.85) baho=5; else
{
if (d>0.71) baho=4; else
{
if (d>0.55) baho=3; else baho=2;
}
}
printf("\n baho=%d", baho);
return 0;
}
```

Kalit bo'yicha tanlash operatori. Kalit bo'yicha tanlash switch operatorining umumiy ko'rinishi quyidagicha:

```
switch(<ifoda>) {
case <1-qiymat>:<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. Uzish break operatori ishlatilmasa, shartga mos kelgan variantdan tashqari keyingi variantdagi operatorlar ham avtomatik bajariladi. Quyidagi default, break va belgilangan variantlar ixtiyoriy tartibda kelishi mumkin. Umuman, 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 <stdio.h>
int main()
{
int baho;
scanf("%d", &baho);
switch(baho)
{
case 2:printf("\n yomon"); break;
case 3:printf("\n o'rta"); break;
case 4:printf("\n yaxshi"); break;
case 5:printf("\n a'lo"); break;
default: printf("\n noto'g'ri kiritilgan");
};
return 0;
}
```

Keyingi misolda kiritilgan simvol unli harf ekanligi aniqlanadi:

```
#include <stdio.h>
int main()
{
char c;
scanf("%c", &c);
switch(c)
{
case 'a':
case 'u':
case 'o':
case 'i':
printf("\n Simvol unli"); break;
default: printf("\n Simvol unli emas");
};
return 0;
}
```

2.3. Sikl operatorlari

Oldingi shartli while operatori. Oldingi shartli while operatori quyidagi umumiy ko'rinishga ega:


```
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 sikl qaytariladi.

Agar dasturda while (1); satr qo'yilsa, bu dastur hech qachon tugamaydi.

Misol. Berilgan n gacha sonlar yig'indisi:

```
#include <stdio.h>
void main()
{
long n, i=1, s=0;
scanf("%d", &n);
while (i<=n)
s+=i++;
printf("\n s=%d", 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:

```
#include <stdio.h>
int main()
{
int nc=0,nl=0;
char c;
while ((c=getchar())!='.')
{
++nc;
if (c== '\n') ++nl;
};
printf("satrlar=%d simvollar=%d \n", nl, nc);
return 0;
}
```

Keyingi shartli do-while operatori. Keyingi shartli do-while operatorining umumiy ko'rinishi quyidagicha:

```
do
Operator
```

while(ifoda)

Sikl 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 sikl qaytariladi.

Misol. Berilgan n gacha sonlar yig'indisi:

```
#include <stdio.h>
int main()
{
long n, i=1, s=0;
scanf("%d", &n);
do
s+=i++;
while (i<=n);
printf("\n s=%d", s);
return 0;
}
```

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

Parametrli for operatori. *Parametrli for operatorining umumiy* ko'rinishi quyidagicha:

for(1-ifoda; 2-ifoda; 3-ifoda)
Operator

Bu operator quyidagi operatorga mos.

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

Misol. Berilgan n gacha sonlar yig'indisi:

```
#include <stdio.h>
int main()
{
int n;
scanf("%d", &n);
int s=0;
for(int i=1; i<=n; i++) s+=i;
printf("\n%d", s);
return 0;
}
```

Siklda bir nechta schotchikning qo'llanilishi. Parametrli for siklining sintaksisi unda bir nechta o'zgaruvchi – schyotchikning qo'llanilishiga, sikl davom etishining murakkab shartlarini tekshirishga va sikl schyotchiklari ustida ketma-ket bir nechta operatsiyaning bajarilishiga imkon beradi.

Agarda bir nechta schyotchikka qiymat o'zlashtirilsa yoki ular o'rtasida bir nechta operatsiya bajarilsa, bu ifodalar vergul bilan ajratilgan holda ketma-ket yoziladi.

for siklida bir nechta schotchikning qo'llanilishi:

```
#include <stdio.h>
#include<conio.h>
int main()
{
int i, j;
for (i=0, j=0; i<3; i++, j++)
printf("i:%d j:%d\n", i, j);
getch();
return 0;
}
```

Natija:

```
i: 0    j: 0
i: 1    j: 1
i: 2    j: 2
```

2.4. O'tish operatorlari

break operatori. Ba'zi hollarda sikl bajarilishini ixtiyoriy joyda to'xtatishga to'g'ri keladi. Bu vazifani break operatori bajarishga imkon beradi. Bu operator darhol sikl bajarilishini to'xtatadi va boshqaruvni sikldan keyingi operatorlarga uzatadi.

Misol:

```
#include <stdio.h>
```

```
int main()
{
int n;
while(1)
{
scanf("%d", &n);
if(n == 1||n == 0) break;
}
printf("Sikl tugadi");
return 0;
}
```

Bu misolda while(1) operatori yordamida cheksiz sikl hosil qilinadi. Agar 1 yoki 0 soni kiritilsa, sikl to'xtatiladi.

continue operatori. Sikl bajarilishiga ta'sir o'tkazishga imkon beradigan yana bir operator continue operatoridir. Bu operator sikl qadamining bajarilishini to'xtatib, for va while da ko'rsatilgan shartli tekshirishga o'tkazadi.

Misol:

```
#include <stdio.h>
int main()
{
int n;
for(;;)
{
scanf("%d", &n);
if(n == 1||n == 0) continue;
break;
}
printf("Sikl tugadi");
return 0;
}
```

Bu misolda for(;;) operatori yordamida cheksiz sikl hosil qilinadi. Agar 1 yoki 0 sonlaridan farqli son kiritilsa, sikl to'xtatiladi.

O'tish operatori goto. O'tish operatorining ko'rinishi:

goto <identifikator>. Bu operator identifikator bilan belgilangan operatorga o'tish kerakligini ko'rsatadi.

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

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

Misol uchun bir necha sikldan birdan chiqish kerak bo'lib qolganda, to'g'ridan to'g'ri break operatorini qo'llab bo'lmaydi, chunki u faqat eng ichki sikldan chiqishga imkon beradi.

```
#include <stdio.h>
int main()
{
int n=16, s=0;
int i, j;
for(i=1; i<5; i++)
for(j=1; j<5; j++)
{
if(i*j>n) goto A;
S++;
}
A:printf("Sikl tugadi s=%d", s);
return 0;
}
```

2.5. Foydalanuvchi funksiyalari

Funksiyalarni ta'riflash va ularga murojaat qilish. Funksiya ta'rifida funksiya nomi, turi va formal parametrlar ro'yxati ko'rsatiladi. Formal parametrlar nomlaridan tashqari turlari ham ko'rsatilishi shart. Formal parametrlar ro'yxati funksiya signaturasi deb ham ataladi.

Funksiya ta'rifining umumiy ko'rinishi quyidagicha:

Funksiya turi funksiya nomi(formal_parametrlar_ta'rifi)

Formal parametrlarga ta'rif berilganda ularning boshlang'ich qiymatlari ham ko'rsatilishi mumkin.

Funksiya qaytaruvchi ifoda qiymati funksiya tanasida **return** <ifoda>; operatori orqali ko'rsatiladi.

Misol:

```
float min(float a, float b)
{
```

```
if (a<b) return a;
return b;
}
```

Funksiyaga murojaat qilish quyidagicha amalga oshiriladi:

Funksiya nomi (haqiqiy parametrlar ro'yxati).

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

Misol uchun yuqoridagi funksiyaga quyidagicha murojaat qilish mumkin:

int x=5, y=6, z; z=min(x, y) yoki **int z=min(5, 6)** yoki **int x=5; int z=min(x, 6)**

Funksiyaga murojaat qilinganda haqiqiy parametrlarning turlari formal parametrlar turlariga mos kelmasligi mumkin. Bu holda avtomatik ravishda turlarni keltirish bajariladi.

Funksiya qiymat qaytarmasa, uning turi **void** deb ko'rsatiladi.

Misol uchun:

```
void print()
{
printf("\n Salom!");
};
```

Bu funksiyaga print() shaklida murojaat qilish ekranga Salom! yozilishiga olib keladi.

Qiymat qaytarmaydigan funksiya tanasida return operatori ishlatilishi mumkin. Bu operator funksiyadan chiqishni bildiradi. Masalan:

```
void print()
{
printf("\n Salom!");
return;
printf("\n Dunyo!");
}
```

Bu funksiyaga print() shaklida murojaat qilish ekranga Salom! yozilishiga olib keladi, lekin **Dunyo!** so'zi yozilmay qoladi.

Qiymat qaytarmaydigan funksiya formal parametrlarga ega bo'lishi mumkin.

Masalan:

```
#include <stdio.h>
void print_baho(int baho)
{
switch(baho)
{
case 2:printf("\n yomon"); break;
case 3:printf("\n o'rta"); break;
case 4:printf("\n yaxshi"); break;
case 5:printf("\n alo"); break;
default:printf("\n noto'g'ri kiritilgan»);
};
};
int main()
{
int a;
scanf("%d", &a);
print_baho(5);
return 0;
};
```

Funksiya prototipi. Agar dasturda funksiya ta'rifi murojaatdan keyin berilsa yoki funksiya boshqa faylda joylashgan bo'lsa, murojaatdan oldin shu funksiyaning prototipi joylashgan bo'lishi kerak. Prototip funksiya nomi va formal parametrlar turlaridan iborat bo'ladi. Formal parametrlar nomlarini berish shart emas.

Misol uchun y=min(a, b)+2*max(c, d) ifodani hisoblashni ko'ramiz:

```
#include <stdio.h>
int max(int a, int b)
{
if (a<b) return b; else return a;
}
int main()
{
int a, b, c, d, y;
int min(int, int);
scanf("%d%d%d%d", &a, &b, &c, &d);
```

```
y=min(a, b)+2*max(c, d);
printf("\n %d", y);
return 0;
};
int min(int a, int b)
{
if (a<b) return a;
else return b;
}
```

Funksiyaga parametrlar uzatish. Funksiyaga parametrlar qiymat bo'yicha uzatiladi va quyidagi bosqichlardan iborat bo'ladi:

1. Funksiya bajarishga tayyorlanganda formal parametrlar uchun xotiradan joy ajratiladi, ya'ni formal parametrlar funksiyalarning ichki parametrlariga aylantiriladi. Agar parametr turi float bo'lsa, double turidagi obyektlar hosil bo'ladi, char va shortint bo'lsa, int turidagi obyektlar yaratiladi.

2. Haqiqiy parametrlar sifatida ishlatilgan ifodalar qiymatlari hisoblanadi.

3. Haqiqiy parametrli ifodali qiymatlari formal parametrlar uchun ajratilgan xotira qismlariga yoziladi. Bu jarayonda float turi double turiga, char va short int turlari int turiga keltiriladi.

4. Funksiya tanasi ichki obyektlar – parametrlar yordamida bajariladi va qiymat chaqirilgan joyga qaytariladi.

5. Haqiqiy parametrlar qiymatlariga funksiya hech qanday ta'sir o'tkazmaydi.

6. Funksiyadan chiqishda formal parametrlar uchun ajratilgan xotira qismlari bo'shatiladi.

C tilida chaqirilgan funksiya chaqiruvchi funksiyadagi o'zgaruvchi qiymatini o'zgartira olmaydi. U faqat o'zining vaqtinchalik nusxasini o'zgartirishi mumkin xolos.

Qiymat bo'yicha chaqirish qulaylik tug'diradi. Chunki funksiyalarda kamroq o'zgaruvchilarni ishlatishga imkon beradi. Misol uchun shu xususiyatni aks ettiruvchi POWER funksiyasi variantini keltiramiz:

```
int power(int x, int 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. Parametr n funksiya ichida o'zgarishi funksiyaga murojaat qilingan boshlang'ich qiymatiga ta'sir qilmaydi.

2.6. Rekursiya

Rekursiv funksiyalar. Rekursiv funksiya deb o'ziga o'zi murojaat qiluvchi funksiyaga aytiladi. Misol uchun faktorialni hisoblash funksiyasini keltiramiz:

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

Manfiy argument uchun funksiya 0 qiymat qaytaradi. Parametr 0 ga teng bo'lsa, funksiya 1 qiymat qaytaradi. Aks holda parametr qiymati birga kamaytirilgan holda funksiyaning o'zi chaqiriladi va uzatilgan parametrga ko'paytiriladi. Funksiyaning o'z-o'zini chaqirish formal parametr qiymati *0 ga teng bo'lganda to'xtatiladi.*

Keyingi misolimizda ixtiyoriy haqiqiy sonning butun darajasini hisoblash rekursiv funksiyasini 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 funksiyaga expo(2.0,3) shaklda murojaat qilinganda rekursiv ravishda funksiyaning ikkinchi parametri kamaygan holda murojaatlar

hosil bo'ladi: expo(2.0,3), expo(2.0,2), expo(2.0,1), expo(2.0,0). Bu murojaatlarda quyidagi ko'paytma hisoblanadi: 2.0*2.0*2.0*1 va kerakli natija hosil qilinadi.

Shuni ko'rsatib o'tish kerakki, bu funksiyamizda noaniqlik mavjuddir, ya'ni 0.0 ga teng sonning 0-darajasi 0 ga teng bo'ladi. Matematik nuqta nazardan bo'lsa, bu holda noaniqlik kelib chiqadi. Yuqoridagi sodda misollarda rekursiyasiz iterativ funksiyalardan foydalanish maqsadga muvofiq.

Masalan, darajani hisoblash funksiyani quyidagicha tuzish mumkin:

```
#include <stdio.h>
double expo(double a, int n)
{
if (n == 0) return 1;
if (a == 0.0) return 0;
int k=(n>0)?n:-n;
double s=1.0;
for(int i=0; i<k; i++) s*=a;
if (n>0) return s; else return 1/s;
}
void main()
{
printf("%f", expo(2, –2));
};
```

Natija

```
0.250000
```

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

Rekursiv usulda funksiya har bir chaqiriqda bosh raqamlardan nusxa olish uchun o'z-o'ziga murojaat qiladi, so'ngra oxirgi raqamni bosib chiqaradi.

```
#include <stdio.h>
void printd(int n){
int i;
if (n<0) {
```



```
printf("–");
n=–n;
}
if ((i=n/10)!=0)
printd(i);
printf("%d", n % 10);
}
void main()
{
printd(123);
};
```

Bu misolda printd (123) chaqiriqda birinchi funksiya printd n=123 qiymatga ega. U 12 qiymatni ikkinchi printd ga uzatadi, boshqarish o'ziga qaytganda, 3 ni chiqaradi.

2.7. Razryadli arifmetika

Maska. Maska deb tanlangan bitlari 1 ga teng bo'lib, qolgan bitlari 0 ga teng bo'lgan songa aytiladi. Sodda maska deb bitta tanlangan bit 1 ga teng bo'lib, qolgan bitlari 0 ga teng bo'lgan songa aytiladi. Ta'rifdan ko'rinib turibdiki, sodda maska ikkining darajalaridan iboratdir.

Masalan, uchinchi biti noldan farqli maska hosil qilish:

```
unsigned mask=1;
mask<<=2;
```

Maskalar ko'pincha & amali bilan qo'llanadi:

```
unsigned mask=2;
val&=mask
```

Bu misolda val o'zgaruvchining oxiridan ikkinchi bitidan tashqari hamma bitlar 0 ga teng bo'ladi. Oxiridan ikkinchi biti o'zgarmaydi.

Yana bir misol:

```
ch &=0xff; /* yoki ch &=0377; */
```

Ma'lumki, 0xff qiymatining binar ko'rinishi 11111111, ya'ni 0377. Bu maska o'zgaruvchi oxirgi 8 bitini o'zgartirmasdan, qolganlarini 0 ga o'rnatadi.

Bitni 1 ga o'rnatish:

```
unsigned mask=2;
flags|=MASK;
```

Bu misolda oxiridan ikkinchi bit 1 ga o'rnatiladi. Qolgan bitlar qiymatlari o'zgarmaydi.

Bitni 0 ga o'rnatish:

```
unsigned mask=2;
flags &=~MASK;
```

Bu misolda oxiridan ikkinchi bit 0 ga o'rnatiladi, qolgan bitlar qiymatlari o'zgarmaydi.

Bitni teskariga o'rnatish:

```
unsigned mask=2;
flags^=mask;
```

Bu misolda oxiridan ikkinchi bit 0 bo'lsa, 1 ga o'rnatiladi va 1 bo'lsa, 0 ga o'rnatiladi. Qolgan bitlar qiymatlari o'zgarmaydi.

Hamma sodda maskalarni chiqarish:

```
#include<stdio.h>
#include<conio.h>
int main()
{
unsigned mask=1;
int i;
for(i=1; i<33; i++)
{
printf("\n %d %u", i, mask);
mask<<=1;
}
getch();
return 0;
}
```

Hamma bitlarni chiqarish. Butun ishorali sonlar tasvirlanganda yetakchi bit ishorani ko'rsatish uchun ishlatiladi. Musbat butun sonlar to'g'ri kodda tasvirlanadi, ya'ni bosh kodi nolga teng bo'lib, qolgan bitlar sonning ikkilik ko'rinishi asosida tasvirlanadi. Masalan, 3 kodi 32 bitli tasviri: 00000000000000000000000000000011.

Manfiy sonlar qo'shimcha kodda tasvirlanadi, ya'ni musbat soni tasviridagi hamma kodlar teskarisiga aylantiriladi. So'ngra 1 qo'shiladi. Masalan, –3 kodining 32 bitli tasviri: 11111111111111111111111111111101.

Sonning hamma bitlarini chiqarish uchun siklda 1 soni bilan & amali natijasi ekranga chiqariladi va songa surish >> amali qo'llaniladi.

Dastur:

```
#include<stdio.h>
void printbits(int c)
{
int i;
unsigned k=8*sizeof(int);
for(i=0; i<k; i++)
{
printf("%d", c&1);
c>>=1;
}
}
int main()
{
printbits(5);
return 0;
}
```

Bu dastur natijasida son bitlari ekranga teskari tartibda chiqadi. To'g'ri tartibda chiqarish uchun sonning o'zini avval aylantirib olinadi. Quyidagi dasturda sonni aylantirish va bitlarni ekranga chiqarish funksiyalaridan foydalanilgan:

```
#include<stdio.h>
int reversebits(int c)
{
int i;
int m=0;
int k;
unsigned l=8*sizeof(int);
for(i=0; i<l; i++)
{
k=c&1;
m<<=1;
m|=k;
```

```
c>>=1;
}
return m;
}
void printbits(int c)
{
int i;
unsigned k=8*sizeof(int);
for(i=0; i<k; i++)
{
printf("%d", c&1);
c>>=1;
}
}
int main()
{
int m;
m=reversebits(-3);
printf("%u\n", m);
printbits(m);
return 0;
}
```

Keyingi misolimizda berilgan sonning n bitini o'zgartirish funksiyasi yaratilgan. Ma'lumki, operator ~ hamma bitlarni teskariga aylantiradi. Ma'lum bitlarni tanlashga imkon bermaydi. Quyida maskadan foydalanib, n bitni teskariga aylantirish ko'rsatilgan:

```
#include<stdio.h>
int invert_end(int num, int bits)
{
int mask=0;
int bitval=1;
while (bits-- > 0)
{
mask |=bitval;
bitval <<=1;
}
```



```
return num ^ mask;
}

void printbits(int c)
{
int i;
unsigned k=8*sizeof(int);
for(i=0; i<k; i++)
{
printf("%d", c&1);
c>>=1;
}
}
int main()
{
int m;
printbits(5);
printf("\n");
m=invert_end(5,3);
printbits(m);
return 0;
}
```

2-bob bo'yicha savollar

1. Sikl while operatorining umumiy ko'rinishi.
2. Sikl do while operatori while operatoridan qanday farq qiladi?
3. Sikl for operatorining umumiy ko'rinishi.
4. Sirpanish deb nimaga aytiladi?
5. Continue operatoridan nima maqsadda foydalaniladi?
6. Break operatoridan nima maqsadda foydalaniladi?
7. O'tish goto operatori boshqarishni qayerga uzatadi?
8. Funksiya prototipini e'lon qilish va funksiyani aniqlash o'rtasida qanday farq bor?
9. Agarda funksiya hech qanday qiymat qaytarmasa, uni qanday e'lon qilish kerak?
10. Rekursiya nima?

1. Berilgan eps aniqlikda umumiy hadi 1/n! bo'lgan ketma-ketlik yig'indisini hisoblovchi dastur tuzing.
2. Umumiy hadi x/n! bo'lgan ketma-ketlik n ta hadining yig'indisini hisoblovchi dastur tuzing.
3. Kiritilgan n ta son qat'iy o'suvchi ekanligini tekshiruvchi dastur yarating.
4. Kiritilgan n simvoldan nechtasi unli harf ekanligini switch operatori yordamida hisoblovchi dastur tuzing.
5. Rekursiya yordamida Paskal uchburchagini hisoblovchi funksiya tuzing. Bu funksiya yordamida uchburchakni ekranga chiqaruvchi funksiya tuzib, dasturda foydalaning.

3-bob. MASSIVLAR VA SATRLAR

3.1. Bir o'lchovli massivlar

Massiv tushunchasi. Massiv – bu bir turli raqamlangan ma'lumotlar jamlanmasidir. Massiv indeksli o'zgaruvchi tushunchasiga mos keladi. Massiv ta'riflanganda turi, nomi va indekslar chegarasi ko'rsatiladi. Masalan, type turidagi length ta elementdan iborat a nomli massiv shunday e'lon qilinadi:

```
type a[length];
```

Bu maxsus a[0], a[1], ..., a[length –1] nomlarga ega bo'lgan type turidagi o'zgaruvchilarning e'lon qilinishiga to'g'ri keladi.

Massivning har bir elementi o'z raqamiga – indeksga ega. Massivning x-elementiga murojaat indekslash operatsiyasi yordamida amalga oshiriladi:

```
int x=...; //butun sonli indeks
TYPE value=a[x]; //x-elementni o'qish
a[x]=value; //x-elementga yozish
```

Indeks sifatida butun tur qiymatini qaytaradigan har qanday ifoda qo'llanishi mumkin: char, short, int, long. C da massiv elementlarining indekslari 0 dan boshlanadi (1 dan emas), length elementdan iborat bo'lgan massivning oxirgi elementining indeksi esa – bu length – 1 (length emas)ga teng. Massivning int z[3] shakldagi ta'rifi int turiga tegishli z[0], z[1], z[2] elementlardan iborat massivni aniqlaydi.

Massiv chegarasidan tashqariga chiqish (ya'ni mavjud bo'lmagan elementni o'qish/yozishga urinish) dastur bajarilishida kutilmagan natijalarga olib kelishi mumkin. Shuni ta'kidlab o'tish lozimki, bu eng ko'p tarqalgan xatolardan biridir.

Agar massiv initsializatsiya qilinganda elementlar chegarasi ko'rsatilgan bo'lsa, ro'yxatdagi 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. Agar massiv uzunligiga qaraganda kamroq element berilgan bo'lsa, qolgan elementlar 0 hisoblanadi:

int a10[10] = {1, 2, 3, 4}; //va 6 ta nol

Agar nomlangan massivning tavsifida uning o'lchamlari ko'rsatilmagan bo'lsa, kompilator tomonidan massiv chegarasi avtomatik aniqlanadi:

int a3[] = {1, 2, 3};

Massivda musbat elementlar soni va summasini hisoblash:

```
#include<stdio.h>
int main()
{
int s=0, k=0;
int x[ ]={–1, 2, 5, –4, 8, 9};
for(int i=0; i<6; i++)
{
if (x[i]<=0) continue;
k++;
s+=x[i];
};
printf("%d\n", k);
printf("%d\n", s);
return 0;
};
```

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

```
#include<stdio.h>
int main()
{
int i, j, n;
float min, max, s=0;
float a, b, d, x[100];
while(1)
{
printf("\n n=");
scanf("%d", &n);
if (n>0 && n<=100) break;
printf("\n Xato 0<n<101 bo'lishi kerak");
}
printf("\n elementlar qiymatlarini kiriting:\n");
for (i=0; i<n; i++)
{
printf("x[%d]=", i);
scanf("%f", &x[i]);
}
max=x[0];
min=x[0];
for(i=0; i<n; i++)
{
s+=x[i];
if (max<x[i]) max=x[i];
if (min>x[i]) min=x[i];
};
s/=n;
printf("\n max=%f", max);
printf("\n min=%f", min);
printf("\n o'rta qiymat=%f", s);
return 0;
};
```

Massivlarni navlarga ajratish. Navlarga ajratish – bu berilgan ko'plab obyektlarni biror-bir belgilangan tartibda qaytadan guruhlash jarayoni.

Massivlarning navlarga ajratilishi ularning bajarilish tezligiga ko'ra farqlanadi. Navlarga ajratishning n*n ta qiyoslashni talab qilgan oddiy usuli va n*log(n) ta qiyoslashni talab qilgan tez usuli mavjud. Oddiy usullar navlarga ajratish tamoyillarini tushuntirishda qulay hisoblanadi, chunki sodda va kalta algoritmlarga ega. Murakkablashtirilgan usullar kamroq sonli operatsiyalarni talab qiladi, biroq operatsiyalarning o'zi murakkabroq, shuning uchun uncha katta bo'lmagan massivlar uchun oddiy usullar ko'proq samara beradi.

Oddiy usullar uchta asosiy kategoriyaga bo'linadi:

– oddiy kiritish usuli bilan navlarga ajratish;
– oddiy ajratish usuli bilan navlarga ajratish;
– oddiy almashtirish usuli bilan navlarga ajratish.

Oddiy kiritish usuli bilan navlarga ajratish

Massiv elementlari avvaldan tayyor berilgan va dastlabki ketma-ketliklarga bo'linadi. i=2 dan boshlab, har bir qadamda dastlabki ketma-ketlikdan i- element chiqarib olinadi hamda tayyor ketma-ketlikning kerakli o'rniga kiritib qo'yiladi. Keyin i bittaga ko'payadi va h.k.

Kerakli joyni izlash jarayonida, ko'proq, o'ngdan bitta pozitsiyadan tanlab olingan elementni uzatish amalga oshiriladi, ya'ni tanlab olingan element, j=i–1 dan boshlab, navlarga ajratib bo'lingan qismning navbatdagi elementi bilan qiyoslanadi. Agar tanlab olingan element a[i] dan katta bo'lsa, uni navlarga ajratish qismiga qo'shadilar, aks holda a[j] bitta pozitsiyaga suriladi, tanlab olingan elementni esa navlarga ajratilgan ketma-ketlikning navbatdagi elementi bilan qiyoslaydilar. To'g'ri keladigan joyni qidirish jarayoni ikkita turlicha shart bilan tugallanadi:

agar a[j]>a[i] elementi topilgan bo'lsa;
agar tayyor ketma-ketlikning chap uchida bo'lsa.

```
int i, j, x;
for(i=1; i<n; i++)
{
x=[i]; /* kiritib qo'yishimiz lozim bo'lgan elementni esda saqlab qolamiz */
j=i-1;
while(x<a[j]&&j>=0)//to'g'ri keladigan joyni qidirish
{
a[j+1]=a[j]; //o'ngga surish
j--;
}
a[j+1]=x; //elementni kiritish
}
```

Oddiy tanlash usuli bilan navlarga ajratish

Massivning minimal elementi tanlanadi hamda massivning birinchi elementi bilan joy almashtiriladi. Keyin jarayon qolgan elementlar bilan takrorlanadi va h.k.


```
int i, min, n_min, j;
for(i=0; i<n-1; i++)
{
min=a[i]; n_min=i; //minimal qiymatni qidirish
for(j=i+1; j<n; j++)
if(a[j]<min){min=a[j]; n+min=j;}
a[n_min]=a[i]; //almashtirish
a[i]=min;
}
```

Oddiy almashtirish usuli bilan navlarga ajratish

Elementlar juftlari oxirgisidan boshlab qiyoslanadi va o'rin almashinadi. Natijada massivning eng kichik elementi uning eng chapki elementiga aylanadi. Jarayon massivning qolgan elementlari bilan davom ettiriladi.

```
for(int i=1; i< n; i++)
for(int j=n-1; j >=i; j-
if(a[j]<a[j-1])
{int r=a[j]; a[j]=a[j-1]; a[j-1]=r;}
}
```

Bir o'lchamli massivlarni funksiya parametrlari sifatida uzatish. Massivdan funksiya parametri sifatida foydalanganda, funksiyaning birinchi elementiga ko'rsatkich uzatiladi, ya'ni massiv hamma vaqt adres bo'yicha uzatiladi. Bunda massivdagi elementlarning miqdori haqidagi axborot yo'qotiladi, shuning uchun massivning o'lchamlari haqidagi ma'lumotni alohida parametr sifatida uzatish kerak.

Misol:

Massivning barcha elementlari chiqarilsin:

```
#include <stdio.h>
#include <stdlib.h>
int form(int a[100])
{
int n;
printf("\nEnter n:");
scanf("%d", &n);
for(int i=0; i<n; i++)
a[i]=rand()%100;
```

```
return n;
}
void print(int a[100], int n)
{
for(int i=0; i<n; i++)
printf("%d", a[i]);
printf("\n");
}
int main()
{
int a[100];
int n;
n=form(a);
print(a, n);
return 0;
}
```

Funksiyaga massiv boshlanishi uchun ko'rsatkich uzatilgani tufayli (adres bo'yicha uzatish), funksiya tanasining operatorlari hisobiga massiv o'zgarishi mumkin.

Funksiyalarda bir o'lchovli sonli massivlar argument sifatida ishlatilganda ularning chegarasini ko'rsatish shart emas.

Misol. Massivdan barcha juft elementlar chiqarilsin.

```
#include <stdio.h>
#include <stdlib.h>
void form(int a[], int n)
{
for(int i=0; i<n; i++)
a[i]=rand()%100;
}
void print(int a[], int n)
{
for(int i=0; i<n; i++)
printf("%d", a[i]);
printf("\n");
}
```



```
int Dell(int a[], int n)
{
int j=0, i, b[100];
for(i=0; i<n; i++)
if(a[i]%2!=0)
{
b[j]=a[i]; j++;
}
for(i=0; i<j; i++) a[i]=b[i];
return j;
}
int main()
{
int a[100];
int n;
printf("\nEnter n:");
scanf("%d", &n);
form(a, n);
print(a, n);
n=Dell(a, n);
print(a, n);
system("pause");
return 0;
}
```

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

Vektorning uzunligini aniqlash funksiyasi:

```
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));
}
```

Funksiyalarda bir o'lchovli massivlar qaytariluvchi qiymatlar sifatida ham kelishi mumkin. Misol uchun ikki vektor yig'indisini hisoblovchi funksiyani 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 funksiyaga quyidagicha murojaat qilish mumkin:

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

Polinom. Polinom qiymatini hisoblash funksiyasi poly deb nomlanadi.

Prototipi:

double poly(double x, int degree, double coeffs[]);

Algebraik polinom koeffitsiyentlari coeffs[0], coeffs[1], ..., coeffs [degree] massiv elementlarida beriladi.

Misol:

```
#include <stdio.h>
#include <math.h>
/* polynomial: x**3–2x**2+5x–1 */
int main(void)
{
double array[]={–1.0, 5.0, –2.0, 1.0};
double result;
result=poly(2.0, 3, array);
printf("The polynomial: x**3–2.0x**2+5x–1 at 2.0 is %lf\n", result);
return 0;
}
```

3.2. Ko'p o'lchovli massivlar

Ko'p o'lchovli massivlar ta'rifi. Ikki o'lchovli massivlar matematikada matrisa yoki jadval tushunchasiga mos keladi. Jadvallarning initsializatsiya qilish qoidasi, ikki o'lchovli massivning elementlari massivlardan iborat bo'lgan bir o'lchovli massiv ta'rifiga asoslangan.

Misol uchun ikki qator va uch ustundan iborat bo'lgan haqiqiy turga tegishli d massivning boshlang'ich qiymatlari quyidagicha ko'rsatilishi mumkin:

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

Bu yozuv quyidagi qiymat berish operatorlariga mos:

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'yxat 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 quyidagi qiymat berish operatorlariga mos:

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.

Quyidagi ko'radigan misolimizda jadval kiritilib, har bir qatorning maksimal elementi aniqlanadi:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
int a[4][3];
int max;
int i, j;
for(i=0; i<4; i++)
{
for(j=0; j<3; j++)
{
printf("a[%d][%d]=", i, j);
scanf("%d", &a[i][j]);
}
```

```
printf("\n");
};
for(i=0; i<4; i++)
{
max=a[i][0];
for(j=0; j<3; j++)
if (max<a[i][j]) max=a[i][j];
printf("%d", max);
}
system("pause");
return 0;
}
```

Funksiyaga ko'p o'lchamli massivlarni uzatish. Ko'p o'lchamli massivlarni funksiyaga uzatishda barcha o'lchamlar parametrlar sifatida uzatilishi kerak. C tilida ko'p o'lchamli massivlar aniqlanishi bo'yicha mavjud emas. Agar biz bir nechta indeksga ega bo'lgan massivni tavsiflasak (masalan, int mas [3][4]), bu degani, biz bir o'lchamli mas massivini tavsifladik, bir o'lchamli int [4] massivlar esa uning elementlaridir.

Misol: Kvadrat matritsani uzatish (transportirovka qilish).

Agar void transp(int a[][], int n){...} funksiyasining sarlavhasini aniqlasak, bu holda biz funksiyaga noma'lum o'lchamdagi massivni uzatishni xohlagan bo'lib qolamiz. Aniqlanishiga ko'ra, massiv bir o'lchamli bo'lishi hamda uning elementlari bir xil uzunlikda bo'lishi kerak. Massivni uzatishda uning elementlarining o'lchamlari haqida ham biron narsa deyilmagan, shuning uchun kompilator xato chiqarib beradi.

Bu muammoning eng sodda yechimi funksiyani quyidagicha aniqlashdir:

void transp(int a[][4], int n){...}, bu holda har bir satr o'lchami 4 bo'ladi, massiv ko'rsatkichlarining o'lchami esa hisoblab chiqariladi:

```
#include <stdio.h>
#include <stdlib.h>
const int N=4;//global o'zgaruvchi
void input_array(int a[][N], int n)
{
int i, j;
```



```
for(i=0; i<n; i++)
{
for(j=0; j<n; j++)
{
printf("a[%d][%d]=", i, j);
scanf("%d", &a[i][j]);
}
printf("\n");
}
}
void print_array(int a[][N], int n)
{
int i, j;
for(i=0; i<n; i++)
{
for(j=0; j<n; j++)
{
printf("a[%d][%d]=", i, j);
printf("%d", a[i][j]);
}
printf("\n");
}
}
void transp(int a[][N], int n)
{
int r;
for(int i=0; i<n; i++)
for(int j=0; j<n; j++)
if(i<j)
{
r=a[i][j]; a[i][j]=a[j][i]; a[j][i]=r;
}
}
int main()
{
int mas[N][N];
```

```
int n;
printf("\n n=");
scanf("%d", &n);
input_array(mas, n);
transp(mas, n);
print_array(mas, n);
system("pause");
return 0;
}
```

Misol tariqasida uch o'lchovli kvadrat matritsani uch o'lchovli vektorga ko'paytirish funksiyasini ko'rib chiqamiz:

```
void umn_vec(float a[3][3], float b[3], float c[3])
{
for(int i=0; i<3; i++)
{
c[i]=0;
for(int j=0; j<3; j++)
c[i]+=a[i][j]*b[j];
};
}
```

3.3. Belgili axborot va satrlar

Satrlar. C da belgili ma'lumotlar uchun char turi qabul qilingan. Belgili axborotni taqdim etishda belgilar, simvolli o'zgaruvchilar va matnli konstantalar qabul qilingan.

Misollar:

```
const char c='c';
char a, b;
```

C dagi satr – bu nol-belgi – \0 (nol-terminator) – bilan tugallanuvchi belgilar massivi. Nol-terminatorning holatiga qarab satrning amaldagi uzunligi aniqlanadi. Bunday massivdagi elementlar soni, satr tasviriga qaraganda, bittaga ko'p.

Simvolli massivlar quyidagicha initsializatsiya qilinadi:

char capital[]="TASHKENT"; Bu holda avtomatik ravishda massiv elementlari soni aniqlanadi va massiv oxiriga satr ko'chirish '\0' simvoli qo'shiladi.

Yuqoridagi initsializatsiyani quyidagicha amalga oshirish mumkin:

char capital[]={'T','A','S','H','K','E','N','T','\0'};

Bu holda so'z oxirida '\0' simvoli aniq ko'rsatilishi shart.

Qiymat berish operatori yordamida satrga qiymat berish mumkin emas. Satrni massivga yoki kiritish paytida, yoki nomlantirish yordamida joylashtirish mumkin:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
char s1[10]="string1";
int k=sizeof(s1);
printf("\n%s %d", s1, k);
char s2[]="string2";
k=sizeof(s2);
printf("\n%s %d", s2, k);
char s3[]={'s', 't', 'r', 'i', 'n', 'g', '3', '\0'};
k=sizeof(s3);
printf("\n%s %d", s3, k);
char *s4="string4";//satr ko'rsatkichi, uni o'zgartirib bo'lmaydi
k=sizeof(s4);
printf("\n%s %d", s4, k);
system("pause");
return 0;
}
```

Natija:

```
string1 10
string2 8
string3 8
string4 4
```

Keyingi misolda kiritilgan so'zdan berilgan harfni olib tashlash dasturi berilgan.

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

```
char s[100];
scanf("%s", &s);
int i, j;
for (i=j=0; s[i] !='\0'; i++)
if (s[i] !='c')
s[j++]=s[i];
s[j]='\0';
printf("%s", s);
return 0;
}
```

Har safar 's' dan farqli simvol uchraganda, u j pozitsiyaga yoziladi va faqat shundan so'ng j ning qiymati 1 ga oshadi. Bu quyidagi yozuvga ekvivalent:

```
if (s[i] !=c)
s[j]=s[i];
j++;
```

Funksiyalar va satrlar. Funksiyalarda satrlar ishlatilganda ularning chegarasini ko'rsatish shart emas. Satrlarning uzunligini hisoblovchi len funksiyasini quyidagicha ta'riflash mumkin:

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

Shu funksiyadan foydalanilgan dasturni keltiramiz:

```
#include <stdio.h>
int len(char c[])
{
int m=0;
while(c[m++]);
return m-1;
};
int main()
{
```



```
char c[]="Pro Tempore!";
printf("\n%d", len(c));
return 0;
}
```

Bu funksiyaning standart varianti strlen deb ataladi va bu funksiyadan foydalanish uchun string.h sarlavha faylidan foydalanish lozim.

Satrdan nusxa olish funksiyasi strcpy ni C tilida quyidagicha ta'riflash mumkin:

```
#include <stdio.h>
void strlen(char s1[], char s2[])
{
int i=0;
while(s2[i]!='\0') s1[i++]=s2[i];
s1[i]=s2[i];
}
int main()
{
char s1[]="aaa";
char s2[]="ddd";
strcpy(s1, s2);
printf("%s", s1);
return 0;
}
```

Natija:

ddd

Berilgan satrni teskariga aylantiruvchi funksiya:

```
reverse(char s[])
{
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 oxiriga ulovchi STRCAT(S, T) funksiyasini ko'rib chiqamiz:

```
strcat(char s[], t[])
{
int i, j;
i=j=0;
while (s[i] != '\0')
i++;
while((s[i++]=t[j++]) != '\0')
}
```

3.4. So'zlar massivlari

So'zlar massivini kiritish. 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 quyidagicha initsializatsiya qilinishi mumkin:

char Name[3][8] = {"Anvar", "Mirkomil", "Yusuf"}.

Bu ta'rifda har bir so'z uchun xotiradan 8 bayt joy ajratiladi va har bir so'z oxiriga '\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] = {"kompyuter", "printer", "kartridj"}.

Quyidagi dasturda berilgan harf bilan boshlanuvchi so'zlar ro'yxati bosib chiqariladi:

```
#include <stdio.h>
int main()
{
char a[10][10];
char c='a';
int i;
for (i=0; i<3; i++) scanf("%s", &a[i]);
for (i=0; i<3; i++)
if (a[i][0] == c) printf("\n%s", a[i]);
return 0;
}
```


Quyidagi dasturda fan nomi, talabalar ro'yxati va ularning baholari kiritiladi. Dastur bajarilganda ikki olgan talabalar ro'yxati bosib chiqariladi:

```
#include <stdio.h>
int main()
{
char a[10][10];
char s[10];
int k[10];
scanf("%s", &s);
for (int i=0; i<3; i++)
{
scanf("%s", &a[i]);
scanf("%d", &k[i]);
};
for (int i=0; i<3; i++)
if (k[i] == 2) printf("%s\n", a[i]);
return 0;
}
```

Funksiyalar va so'zlar massivlari. Satrli massivlar funksiya argumenti sifatida ishlatilganda satrlarning umumiy uzunligi aniq ko'rsatilishi shart.

Misol tariqasida ixtiyoriy sondagi satrlar massivini alifbo bo'yicha tartiblash funksiyasidan foydalanilgan dasturni ko'rib chiqamiz:

```
#include <stdio.h>
#define m 10
void sort(int n, char a[][m])
{
char c;
int i, j, 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;
};
};
int main()
{
char aa[][m]={"Alimov", "Dadashev", "Boboyev"};
sort(3, aa);
for(int i=0; i<3; i++) printf("%s\n", aa[i]);
return 0;
}
```

3.5. Izlash va tartiblash

Ikkiga bo'lib izlash. Quyidagi dasturda tartiblangan massivda ikkiga bo'lib kalit sonni izlash algoritmi asosida tuzilgan funksiyadan foydalanish keltirilgan:

```
#include<stdio.h>
#include<conio.h>
int bsearch(int a[], int key, int n)
{
int m1, m2, m;
m1=0; m2=n-1;
while(m1<=m2)
{
m=(m2+m1)/2;
if (a[m]==key) return m;
if (a[m]>key) m2=--m;
if (a[m]<key) m1=++m;
}
return -1;
};
int main(int argc, char* argv[])
{
int m;
```



```
int a[]={5, 6, 9, 11};
m=bsearch(a, 7, 4);
printf("%d", m);
getch();
return 0;
}
```

Keyingi misolda shu funksiyaning satrlar uchun varianti keltirilgan:

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
#define size 5
int strbsearch(char a[][size], char key[], int n)
{
int m1, m2, m, pr;
m1=0; m2=n-1;
while(m1<=m2)
{
m=(m2+m1)/2;
pr=strcmp(a[m], key);
if (pr==0) return m;
if (pr==-1) m2=--m;
if (pr==1) m1=++m;
}
return -1;
};
int main(int argc, char* argv[])
{
int m;
char a[][size]={"aaa", "aab", "aac", "aad"};
m=strbsearch(a, "aab", 4);
printf("%d", m);
getch();
return 0;
}
```

Tezkor tartiblash. Quyidagi dasturda tezkor tartiblash algoritmiga asoslangan funksiyadan foydalanilgan. Algoritmning mohiyati shun-

dan iboratki, avval yetakchi element tanlanadi. Funksiyada yetakchi element sifatida boshlang'ich element tanlanadi. Shundan so'ng, massiv ikki qismga ajratiladi. Yetakchi elementdan kichik elementlar past qismga, katta elementlar yuqori qismga to'planadi. Shundan so'ng, rekursiya asosida algoritm ikkala qismga alohida qo'llanadi.

```
#include<stdio.h>
#include<conio.h>
int a[]={5, 4};
void sqsort(int k1, int k2)
{
if(k1<k2){
int i, j, k;
i=k1; j=k2;
while(i<j)
{
if (a[k1] > a[i]) {i++; continue;}
if (a[k1]<a[j]) {j--; continue;}
k=a[i]; a[j]=a[i]; a[i]=k;
}
k=a[k1]; a[k1]=a[i]; a[i]=k;
sqsort(k1, i);
sqsort(i+1, k2);
};
}
int main(int argc, char* argv[])
{
int i;
sqsort(0, 1);
for(i=0; i<2; i++) printf("%d", a[i]);
getch();
return 0;
}
```

3-bob bo'yicha savollar

1. Satr simvolli massivdan qanday farq qiladi?
2. Bir o'lchovli massivlarni initsializatsiya qilish usullarini ko'rsating.

3. Ko'p o'lchovli massiv ta'rifi xususiyatlarini keltiring.
4. Ko'p o'lchovli massivlar initsializatsiyasi xususiyatlari.
5. Satrlarni initsializatsiya qilish usullarini ko'rsating.
6. So'zlar massivi qanday kiritiladi?
7. Qanday qilib bir o'lchovli massivlar formal parametrlar sifatida ishlatilishi mumkin?
8. Qanday qilib ko'p o'lchovli massivlar formal parametrlar sifatida ishlatilishi mumkin?
9. Satrni ta'riflash usullari.
10. Satrlar funksiya parametri sifatida.

3-bob bo'yicha topshiriqlar

1. Berilgan massiv qat'iy kamayuvchi ekanligini tekshiruvchi funksiya tuzing va dasturda foydalaning.
2. Matritsani vektorga ko'paytirish funksiyasini tuzing va dasturda foydalaning.
3. Berilgan satr telefon raqami ekanligini aniqlovchi funksiya tuzing va dasturda foydalaning.
4. Berilgan satr o'zgaruvchi ekanligini tekshiruvchi funksiya tuzing va dasturda foydalaning.
5. Berilgan jumladan eng qisqa so'z ajratib oluvchi funksiya tuzing va dasturda foydalaning.

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

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

Strukturali tur ta'riflanganda tur nomi ko'rsatilmay, konkret strukturalar ro'yxati ko'rsatilishi mumkin:

struct
{Elementlar_ta'riflari}
Konkret_strukturalar_ro'yxati.

Quyidagi ta'rif yordamida uchta konkret struktura kiritiladi, lekin strukturali tur kiritilmaydi.

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

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

struct complex sigma = {1.3; 12.6};
struct goods coats = {"pidjak", 40000, 7.5, 220, "12.01.97"};

Bir xil turdagi strukturalarga qiymat berish amalini qo'llash mumkin:

struct complex alfa; alfa = sigma;

Lekin strukturalar uchun solishtirish amallari aniqlanmagan.

Strukturalar elementlariga quyidagicha murojaat qilish mumkin:

Struktura nomi.element_nomi.

'Nuqta amali' struktura elementiga murojaat qilish amali deyiladi. Bu amal qavs amallari bilan birga eng yuqori ustivorlikka ega.

Misol:

struct 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 zarur. Quyidagi misolda xizmatchi strukturasi kiritiladi:

```
#include <stdio.h>
struct employee
{
char name [64];
long employee_id;
float salary;
char phone[10];
int office_number;
} worker;
void show_emploee(employee worker)
{
printf("\nIsmi: %s", worker.name);
printf("\nTelefon: %s", worker.phone);
printf("\nNomer: %ld", worker.employee_id);
printf("\nOylik: %f", worker.salary);
printf("\nOfis: %d", worker.office_number);
};
int main()
{
worker.employee_id = 12345;
worker.salary = 25000.00;
worker.office_number = 102;
printf("\n ismi:");
scanf("%s", &worker.name);
printf("\n telefon:»);
scanf("%s", &worker.phone);
show_emploee(worker);
return 0;
}
```

Strukturaviy turni kiritish. Strukturaviy turni kiritishda yana bir imkoniyatni yordamchi so'z typedef yaratadi. Strukturaviy tur kiritilgan va qayta nomlangan hol uchun ta'rif quyidagi ko'rinishga ega bo'ladi:

typedef struct{elementlar tarifi}

struktura turi

Masalan:

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

Keltirilgan qoida strukturaviy tur va unga belgilash kiritadi. Bu belgilash orqali struktura turidagi o'zgaruvchilarni xuddi shunday oddiy nomlangan strukturaviy tur kabi kiritish mumkin.

Misol: **complex sigma, alfa;**

Strukturaviy turga dasturchi typedef yordamida nom beradi, shu bilan birga, u ikkinchi nomga ham struct yordamchi so'z orqali ega bo'la oladi. Misol tariqasida rasional kasrning strukturaviy turini aniqlashni ko'rib chiqamiz:

```
typedef struct rational_fraction
{
int numerator; /* Surat */
int denominator; /* Maxraj*/
} fraction;
```

bu yerda fraction – strukturaviy turning belgilanishi typedef- yordamida kiritilmoqda.

Strukturaviy turlarni standart ko'rinishda kiritish uchun rational_ fraction ishlatiladi.

4.2. Strukturalar va massivlar

Massivlar strukturalar elementlari sifatida. Massivlarning strukturalar elementi sifatida ishlatilishi hech qanday qiyinchilik tug'dirmaydi. Biz yuqorida simvolli massivlardan foydalanishni ko'rdik.

Quyidagi misolda fazoda berilgan nuqtaviy jismni tasvirlovchi komponentalari jism massasi va koordinatalaridan iborat struktura kiritilgan bo'lib, nuqtaning koordinatalar markazigacha bo'lgan masofasi hisoblangan:

```
#include <stdio.h>
#include <math.h>
struct
```



```
{
double mass;
float coord[3];
} point={12.3, {1.0, 2.0, −3.0}};
int main()
{
int i;
float s=0.0;
for (i=0; i<3; i++)
s+=point.coord[i]*point.coord[i];
printf("\n masofa=%f", sqrt(s));
return 0;
}
```

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

Strukturalar massivlari. Strukturalar massivlari oddiy massivlar kabi tasvirlanadi. Yuqorida kiritilgan strukturali turlar asosida quyidagi strukturalar massivlarini kiritish mumkin:

```
struct goods list[100];
complex set [80];
```

Bu ta'riflarda list va set strukturalar nomlari emas, elementlari strukturalardan iborat massivlar nomlaridir. Konkret strukturalar nomlari bo'lib, set[0], set[1] va hokazolar xizmat qiladi. Konkret strukturalar elementlariga quyidagicha murojaat qilinadi: set[0].real – set massivi birinchi elementining real nomli komponentasiga murojaat.

Quyidagi misolda nuqtaviy jismlarni tasvirlovchi strukturalar massivi kiritiladi va bu nuqtalar sistemasi uchun og'irlik markazi koordinatalari (x_c, y_c, z_c) hisoblanadi. Bu koordinatalar quyidagi formulalar asosida hisoblanadi:

$m=\sum m_i$; $x_c=(\sum x_i m_i)/m$; $y_c=(\sum y_i m_i)/m$; $z_c=(\sum z_i m_i)/m$;

```
#include <stdio.h>
struct particle
{
double mass;
double coord [3];
};
int main()
```

```
{
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;
}
printf("\n Massa markazi koordinatalari:");
for (I=0; I<3; I++)
{
center.coord[I]/=center.mass;
printf("\n Koordinata %d %f", (I+1), center.coord[I]);
}
return 0;
}
```

4.3. Struktura xossalari

Strukturalar va funksiyalar. Strukturalar funksiyalar argumentlari sifatida yoki funksiya qaytaruvchi qiymat sifatida kelishi mumkin. Bundan tashqari funksiya argumenti sifatida struktura turidagi massiv kelishi mumkin.

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 yig'indisini hisoblash funksiyasi:

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



```
return c;
}
```

Misol:

```
#include<stdio.h>
#include<conio.h>
typedef struct
{
char name[20];
int year;
} person;
person old_person(person a[], int n)
{
int i;
person s=a[0];
for(i=1; i<n; i++)
if(a[i].year > s.year) s=a[i];
return s;
}
void print_person(person s)
{
printf("name %s\n", s.name);
printf("year %d", s.year);
}
int main()
{
person a[]={{"smit", 34}, {"bobbi", 45}, {"pit", 56}};
person s=old_person(a, 3);
print_person(s);
getch();
return 0;
}
```

Funksiyada bitta struktura turidagi o'zgaruvchi qiymatini o'zgartirish mumkin emas, lekin massiv elementlari qiymatini o'zgartirish mumkin:

```
#include<stdio.h>
#include<conio.h>
struct goods {
```

```
char name[20];
long price;
float percent;
};
void change_percent(struct goods a[], int n, float percent)
{
int i;
for(i=1; i<n; i++) a[i].percent=percent;
}
void print_goods(struct goods s)
{
printf("%s %ld %f\n", s.name, s.price, s.percent);
}
void all_print(struct goods a[], int n)
{
int i;
for(i=0; i<n; i++) print_goods(a[i]);
};
int main()
{
struct goods a[]={{"smit", 34, 0.5}, {"bobbi", 45, 0.7}, {"pit", 56,
0.8}};
all_print(a, 3);
change_percent(a, 3, 0.5);
all_print(a, 3);
getch();
return 0;
}
```

Strukturalar uchun xotiradan joy ajratish. Struktura uchun ajratilgan joy hajmini quyidagi amallar yordamida aniqlash mumkin:

sizeof (strukturali_tur_nomi);

sizeof (struktura_nomi);

sizeof struktura_nomi.

Oxirgi holda struktura nomi ifoda deb qaraladi. Ifodaning turi aniqlanib, hajmi hisoblanadi.

Misol uchun:

sizeof (struct goods)

sizeof *(tea)*

sizeof coat

Murakkab turlar, ya'ni massivlar va strukturali turlar uchun xotiraga talab ularning ta'rifiga bog'liqdir. Masalan, double array[10] ta'rif xotiradan 10*sizeof bayt joy ajratilishiga olib keladi.

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

Bu ta'rif har bir struct mixture turidagi obyekt xotirada sizeof(int)+sizeof(long)+8*sizeof(char) bayt joy egallashini ko'rsatadi. Obyektning aniq hajmini quyidagi amal hisoblaydi:

sizeof(struct mixture)

Xotirani tekislash. Strukturali tur kiritilishi bu tur uchun xotiradan joy ajratilishiga olib kelmaydi. Har bir konkret struktura (obyekt) ta'riflanganda, shu obyekt uchun elementlar turlariga qarab xotiradan joy ajratiladi. Xotiradan joy zich ajratilganda struktura uchun ajratilgan joy hajmi har bir element uchun zarur bo'lgan xotira hajmlari yig'indisiga teng bo'ladi. Shu bilan birga xotiradan joy zich ajratilmasligi ham mumkin, ya'ni elementlar orasida bo'sh joylar ham qolishi mumkin. Bu bo'sh joy keyingi elementni xotira qismlarining qabul qilingan chegaralari bo'yicha tekislash uchun qoldiriladi. Misol uchun butun turdagi elementlar juft adreslardan boshlansa, bu elementlarga murojaat tezroq amalga oshiriladi:

```
#include <stdio.h>
struct Foo
{
char c;
int i;
char s;
};
int main()
{
printf("Foo hajmi=%d", sizeof(Foo));
```

```
}
```

Natija:

Foo hajmi=12

Konkret strukturalarni joylashuviga ba'zi kompilatorlarda #pragma preprotsessor direktivasi yordamida ta'sir o'tkazish mumkin. Bu direktiva quyidagi shaklda:

pragma pack(n)

Bu yerda n ning qiymati 1, 2 yoki 4 ga teng bo'lishi mumkin.

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

pack(2) – elementlarni so'zlar chegaralari bo'yicha tekislash;

pack(4) – elementlarni ikkilangan so'zlar chegaralari bo'yicha tekislash.

Masalan:

```
#include <stdio.h>
#pragma pack(1)
struct Foo
{
char c;
int i;
char s;
};
int main()
{
printf("Foo hajmi=%d", sizeof(Foo));
return 0;
}
```

Natija:

Foo hajmi=6

4.4. Abstrakt turlarni tasvirlash

Amaliy masalalarni yechishda, shu soha uchun mos bo'lgan ma'lumotlar turlarini aniqlab olish qulay. Dasturda bu turlar strukturali turlar sifatida tasvirlanadi. So'ngra shu tur bilan bog'liq hamma funksiyalarni ta'riflab, kutubxona hosil qilinadi. Misol tariqasida kasrlar bilan bog'liq abstrakt tur kiritamiz. Kasrlar ustida quyidagi funksiyalarni kiritamiz.

input() kasr sonni kiritish;

out() kasr sonni ekranga chiqarish;

add() kasrlarni qo'shish;

sub() kasrlarni ayirish;

mult() kasrlarni ko'paytirish;

divide() kasrlarni bo'lish.

Quyidagi dastur kiritilgan funksiyalar yordamida kasrlar bilan ishlashga misol bo'ladi

```
#include<stdio.h>
#include<stdlib.h>
typedef struct rational_fraction
{
int numerator;
int denominator;
} fraction;
fraction input()
{
int N;
fraction dr;
printf("\n Surat:");
scanf("%d", &dr.numerator);
printf("Maxraj:");
scanf("%d", &N);
if (N == 0)
{
printf("\n Xato! Nol maxraj");
exit(0);
}
dr.denominator=N;
return dr;
}
void out(fraction dr)
{
printf("Kasr");
printf("%d/%d \n", dr.numerator, dr.denominator);
}
fraction add (fraction dr1, fraction dr2)
```

```
{
fraction dr;
dr.numerator=dr1.numerator * dr2.denominator+ dr1.denomina-
tor * dr2.numerator;
dr.denominator=dr1.denominator * dr2.denominator;
return dr;
}
fraction sub (fraction dr1, fraction dr2, fraction * pdr)
{
fraction dr;
dr. numerator=dr1.numerator * dr2.numerator- dr2.numerator *
dr1.denominator;
dr. denominator=dr1.denominator* dr2.denominator;
return dr;
}
fraction mult (fraction dr1, fraction dr2)
{
fraction dr;
dr.numerator=dr1.numerator * dr2.numerator;
dr. denominator=dr1.denominator * dr2.denominator;
return dr;
}
fraction divide (fraction d1, fraction d2)
{
fraction dr;
dr.numerator=d1. numerator * d2. denominator;
dr.denominator=d1. denominator * d2. numerator;
return dr;
}
int main()
{
fraction a, b, c, d;
fraction p;
printf("\n Kasr kiriting");
a=input();
b=input();
c=add(a, b);
```



```
out(c);
c=mult(a, b);
out(c);
c=divide(a, b);
out(c);
return 0;
}
```

4.5. Strukturaning boshqa strukturadan tashkil topishi

Struktura o'zgaruvchisi maydon sifatida. Murakkab strukturalarni hosil qilishda oldin uni tashkil etuvchi oddiyroq strukturalarni e'lon qilib, keyin esa ularni birlashtirish orqali strukturani hosil qilish maqsadga muvofiqdir. Masalan, g'ildirak strukturasi, motor strukturasi, uzatish korobkasi strukturasi va boshqa strukturalarni hosil qilib, keyin esa ularni birlashtirish orqali avtomobil strukturasini qurish qo'yilgan masalani yechishni ancha osonlashtiradi.

Quyidagi misolda oddiy nuqta strukturasi yaratilgan. So'ngra moddiy nuqta yaratilib, uning ichida nuqta sinfiga tegishli o'zgaruvchiga ta'rif berilgan:

```
#include<stdio.h>
typedef struct
{
int x;
int y;
}Point;
typedef struct
{
Point p;
float w;
}FPoint;
int main()
{
FPoint X={10, 20, 1.5};
printf("coord x=%d\n", X.p.x);
printf("coord y=%d\n", X.p.y);
printf("weight w=%f", X.w);
```

```
return 0;
}
```

Masalan, to'g'ri to'rtburchak chiziqlardan tashkil topgan. Chiziq esa ikki nuqta orqali aniqlanadi. Har bir nuqta x va y koordinatalar yordamida aniqlanadi. Quyidagi dasturda to'rtburchak strukturasi ko'rsatilgan. To'g'ri to'rtburchak diagonal bo'yicha ikki nuqta va ikki tomon yordamida aniqlanadi. Shuning uchun oldin har bir nuqtaning x, y koordinatalarini saqlash uchun nuqta strukturasi e'lon qilingan.

Nuqta va to'g'ri to'rtburchakning e'lon qilinishi

```
#include<stdio.h>
typedef struct
{
int x;
int y;
}Point;
Point KPoint(int x1, int y1)
{
Point p;
p.x=x1; p.y=y1;
return p;
}
typedef struct
{
Point p1, p2;
int a, b;
}Rectangle;
Rectangle KRectangle(int x1, int y1, int x2, int y2)
{
Rectangle r;
r.p1=KPoint(x1, y1);
r.p2=KPoint(x2, y2);
r.a=x2-x1;
r.b=y2-y1;
return r;
};
int Per(Rectangle p) {return 2 * (p.a+p.b);}
```



```
int Sq(Rectangle p) {return p.a * p.b;}
int main()
{
Rectangle X;
X=KRectangle(10, 20, 50, 80);
printf("Perimetr=%d\n", Per(X));
printf("Yuza=%d", Sq(X));
return 0;
}
```

Natija:

```
Perimetr=200
Yuza=2400
Perimetr=10
Yuza=6
```

4.6. Birlashmalar

Strukturalarga yaqin tushuncha, bu – birlashma tushunchasidir. Birlashmalar union xizmatchi so'zi yordamida kiritiladi. Misol uchun:

```
union
{
long h;
int i, j;
char c[4]
}UNI;
```

Birlashmalarning asosiy xususiyati shundaki, uning hamma elementlari bir xil boshlang'ich adresga ega bo'ladi.

Birlashmalarning asosiy afzalliklaridan biri, xotira biror qismining qiymatini har xil turdagi qiymat shaklida qarash mumkinligidadir.

Misol uchun quyidagicha birlashma:

```
union
{
float f;
unsigned long k;
char h[4];
}fl;
```

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

Quyidagi dastur yordamida birlashma xususiyatini tekshirish mumkin:

```
#include <stdio.h>
enum paytype{CARD, CHECK};
struct{
paytype ptype;
union{
char card[4];
long check;
};
}info;
int main()
{
info.ptype=CHECK;
info.check=77;
switch (info.ptype)
{
case CARD:printf("\nKarta bilan to'lash:%s", info.card); break;
case CHECK:printf("\nChek bilan to'lash:%ld", info.check);
break;
}
return 0;
}
```

Natija

Chek bilan to'lash: 77

Birlashmalar imkoniyatlarini ko'rsatish uchun bioskey() funksiyasidan foydalanishni ko'rib chiqamiz. Bu funksiya bios.h sarlavhali faylda joylashgan bo'lib, quyidagi prototipga ega:

int bioskey(int);

MS DOS operatsion tizimida ixtiyoriy klavishning bosilishi klaviatura buferiga ma'lumot yozilishiga olib keladi.

Agar funksiyaga bioskey(0) shaklda murojaat qilinsa va bufer bo'sh bo'lsa, biror klavish bosilishi kutiladi, agar bufer bo'sh bo'lmasa, funksiya buferdan ikki baytli kodni o'qib, butun son sifatida qaytaradi. Funk-

siyaga bioskey(0) shaklda murojaat qilinsa va bufer bo'sh bo'lsa, biror klavish bosilishi kutiladi, agar bufer bo'sh bo'lmasa, funksiya buferdagi navbatdagi kodni qaytaradi. Funksiyaga bioskey(1) shaklda murojaat qilish bufer bo'sh yoki bo'shmasligini aniqlashga imkon beradi. Agar bufer bo'sh bo'lmasa, funksiya buferdagi navbatdagi kodni qaytaradi, lekin bu kod buferdan o'chirilmaydi.

Quyidagi dastur buferga kelib tushuvchi kodlarni o'qib, monitorga chiqarishga imkon beradi:

```
#include <stdio.h>
#include <bios.h>
int main()
{
union
{
char hh[2];
int ii;
} cc;
unsigned char scn, asc;
printf("\n\n Ctrl+Z bilan chiqish.");
printf("\n Klavishni bosib, kodini oling. \n");
printf("\n SCAN || ASCII");
printf("\n (10) (16) (10) (16)");
do
{
printf("\n");
cc.ii=bioskey(0);
asc=cc.hh[0];
scn=cc.hh[1];
printf("%4d %3xH || %4d %3xH |", scn, scn, asc, asc);
}
while(asc!=26 || scn!=44);
return 0;
}
```

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

Sikl to 26 ASCII kod va 44 scn kod paydo bo'lmaguncha (Ctrl+Z klavishlari bosilmaguncha) davom etadi.

Razryadli maydonlar. Razryadli maydonlar strukturalar va birlashmalar maydonlarining xususiy holidir. Razryadli maydon ta'riflanganda uning uzunligi bitlarda ko'rsatiladi (butun musbat konstanta).

Misol:

```
#include <stdio.h>
struct
{
int a:8;
int b:6;
} xx={64, 64};
int main()
{
printf("\n%d", xx.a);
printf("\n%d", xx.b);
return 0;
}
```

Natija

64

0

Razryadli maydonlar ixtiyoriy butun turga tegishli bo'lishi mumkin. Razryadli maydonlar adresini olish mumkin emas. Xotirada razryadli maydonlarni joylashtirish kompilator va apparaturaga bog'liq.

Razryadli maydonlar yordamida razryadli massivlar hosil qilish mumkin. Yuqorida ko'rilgan sonning hamma bitlarini chiqarish dasturini quyidagicha yozish mumkin:

```
include<stdio.h>
#include<conio.h>
struct bit
{
unsigned int i:1;
};
unsigned printbits(int c, struct bit pp[])
{
```



```
unsigned int i;
unsigned k=8*sizeof(int);
for(i=0; i<k; i++)
{
pp[i].i=c&1;
printf("%d", c&1);
c>>=1;
}
return k;
}
int main()
{
unsigned int k, i;
struct bit pp[100];
k=printbits(-5, pp);
printf("\n");
for(i=k-1; i > 0; i--)
printf("%d", pp[i].i);
printf("%d", pp[0].i);
getch();
return 0;
}
```

4-bob bo'yicha savollar

1. Strukturaning umumiy ko'rinishi.
2. Strukturalar qanday initsializatsiya qilinadi?
3. Struktura elementlariga qanday murojaat qilinadi?
4. Strukturaga ko'rsatkich ta'rifi.
5. Struktura hajmi qanday o'lchanadi?
6. Strukturalar tarkibidagi massivlar elementlariga qanday murojaat qilinadi?
7. Strukturalar massivi qanday initsializatsiya qilinadi?
8. Birlashmalar xossalarini ko'rsating.
9. Razryadli maydon deb qanday maydonlarga aytiladi?
10. Strukturalar orasidagi munosabatlar.

1. Abiturient (ismi, tug'ilgan yili, yiqqan bali, attestat o'rta bali) strukturasini yarating. Struktura turidagi massiv yarating. Ko'rsatilgan raqamli elementni olib tashlang va ko'rsatilgan familiyali elementdan so'ng element qo'shing.
2. Xodim (ismi, lavozimi, tug'ilgan yili, oylik maoshi) strukturasini yarating. Struktura turidagi massiv yarating. Ko'rsatilgan familiyali elementni olib tashlang va ko'rsatilgan raqamli elementdan so'ng element qo'shing.
3. Mamlakat (nomi, poytaxti, aholi soni, egallagan maydoni) strukturasini yarating. Struktura turidagi massiv yarating. Ko'rsatilgan aholi sonidan kichik bo'lgan elementni olib tashlang va ko'rsatilgan nomga ega bo'lgan elementdan so'ng element qo'shing.
4. Davlat (nomi, davlat tili, pul birligi, valuta kursi) strukturasini yarating. Struktura turidagi massiv yarating. Ko'rsatilgan nomga ega bo'lgan elementni o'chiring va fayl oxiriga ikkita element qo'shing.
5. Inson (ismi, yashash manzili, telefon raqami, yoshi) strukturasini yarating. Struktura turidagi massiv yarating. Ko'rsatilgan yoshga ega bo'lgan elementni o'chiring va berilgan telefon raqamidagi elementdan oldin element qo'shing.

5-bob. KO'RSATKICHLAR, MASSIVLAR, FUNKSIYALAR

5.1. Ko'rsatkichlar

Ko'rsatkichlar. Ko'rsatkich – xotira uyasining unikal adresini saqlaydigan o'zgaruvchi. Ko'rsatkich operativ xotiradagi biror-bir o'zgaruvchi mavjud bo'lishi mumkin bo'lgan biror-bir joyni belgilaydi. Ko'rsatkichlarning qiymatlarini o'zgartirishni shunday turli variantlarda qo'llash mumkinki, bu dasturning moslashuvchanligini oshiradi.

Ko'rsatkich, odatda, turga ega bo'lib, quyidagicha e'lon qilinadi:

<turning nomi>*<ko'rsatkichning nomi>=<dastlabki qiymat>

Misol uchun:

```
int *pr;
char *alfa;
```

Bu holda ko'rsatkichlar noaniq qiymatga ega bo'ladi. Ko'rsatkichlar ta'riflanganda ularning turlari ko'rsatilishi shart. Ko'rsatkichlarni initsializatsiya qilish, ya'ni boshlang'ich qiymatlarini kiritish mumkin. Ma'lum turdagi biror-bir o'zgaruvchi adresi yoki NULL qiymat dastlabki qiymat bo'lishi mumkin. Ko'rsatkichlarga boshlang'ich maxsus NULL qiymati berilsa, bunday ko'rsatkich bo'sh ko'rsatkich deb ataladi.

Biror-bir o'zgaruvchi adresini olish hamda uni ko'rsatkichga qiymat sifatida berish uchun «&» operatori qo'llanadi.

Misol:

```
int I=100;
int*p=&I;
unsigned longint *ul=NULL;
```

Misol:

```
#include <stdio.h>
int main()
{
int i=123;
```

```
int *p=&i;
printf("\n i=%d p=%p", i, p);
int j=456;
p=&j;
printf("\n j=%d p=%p", j, p);
return 0;
}
```

Kelib chiqadi:

i=123 p=0012FF60

j=456 p=0012FF48

Teskari operator – «*» bo'lib, ko'rsatkichda saqlanayotgan adres bo'yicha uya qiymatiga murojaat qilish imkonini beradi.

Misol:

```
int I=100;
int*p=&I
int J=*p;
```

Misol:

```
#include <stdio.h>
int main()
{
int i=123;
int *si=&i;
printf("\n i=%d *si=%d", i, *si);
i=456;
printf("\n i=%d *si=%d", i, *si);
*si=0;
printf("\n i=%d *si=%d", i, *si);
return 0;
}
```

Kelib chiqadi:

i=123 *si=123

i=456 *si=456

i=0 *si=0

5.2. Ko'rsatkichlarning xossalari

Ko'rsatkichlar ustida o'tkaziladigan amallar. Ko'rsatkichlar ustida unar amallar bajarish mumkin: inkrement (++) va dekrement (--) oper-

atsiyalarini bajarishda, ko'rsatkich qiymati ko'rsatkich murojaat qilgan tur uzunligiga ko'payadi yoki kamayadi.

Misol:

```
int*ptr, a[10];
ptr=&a[5];
ptr++; /*=a[6] elementining adresiga */
ptr--; /*=a[5] elementining adresiga */
```

Qo'shish va ayirish binar amallarida ko'rsatkich va int turining qiymati ishtirok etishi mumkin. Bu amal natijasida ko'rsatkich qiymati dastlabkisidan ko'rsatilgan elementlar soniga ko'proq yoki kamroq bo'ladi.

Misol:

```
int*ptr1, *ptr2, a[10];
int i=2;
ptr1=a+(i+4); /*=a[6] elementining adresiga */
ptr2=ptr1–i; /*=a[4] elementining adresiga */
```

Ayirish amalida bitta turga mansub bo'lgan ikkita ko'rsatkich ishtirok etishi mumkin. Amal natijasi int turiga ega hamda kamayuvchi va ayiruvchi o'rtasidagi dastlabki tur elementlarining soniga teng. Bundan tashqari agar birinchi adres kichikroq bo'lsa, u holda natija manfiy qiymatga ega bo'ladi.

Misol:

```
int *ptr1, *ptr2, a[10];
int i;
ptr1=a+4;
ptr2=a+9;
i=ptr1 – ptr2; /*=5 */
i=ptr1 – ptr2; /*=–5 */
```

Bir turga taalluqli bo'lgan ikkita ko'rsatkich qiymatlarini ==, !=, <, <=, >, >=amallari yordamida o'zaro qiyoslash mumkin. Bunda ko'rsatkichlarning qiymatlari shunchaki butun sonlar sifatida olib qaraladi, qiyoslash natijasi esa 0 (yolg'on) yoki 1 (rost)ga teng bo'ladi.

Misol:

```
int *ptr1, *ptr2, a[10];
ptr1=a+5;
ptr2=a+7;
if(ptr1 > ptr2) a[3]=4;
```

Bu misolda ptr1 ning qiymati ptr2 ning qiymatidan kamroq, shuning uchun a[3]=4 operatori bajarilmay qoladi.

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

<tur>* const<ko'rsatkich nomi>=<konstanta ifoda>

Misol uchun:

char* const key_byte=(char*)0x0417.

Bu misolda konstanta ko'rsatkich klaviatura holatini ko'rsatuvchi bayt bilan bog'langan.

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

Konstantaga ko'rsatkich quyidagicha ta'riflanadi:

<tur>const* <ko'rsatkich nomi>=<konstanta ifoda>.

Misol uchun: const int zero=0; int const* p=&zero;

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

<tur>const* const<ko'rsatkich nomi>=<konstanta ifoda>.

Misol uchun:

const float pi=3.141593; float const* const pp=π

Turlashtirilmagan ko'rsatkich. Turlashtirilmagan (tipiklashtirilmagan) ko'rsatkich void turga ega bo'lib, ixtiyoriy turdagi o'zgaruvchi adresi qiymat sifatida berilishi mumkin.

Maxsus void turidagi ko'rsatkichlar ajdodiy ko'rsatkichlar deb atalib, har xil turdagi obyektlar bilan bog'lanish uchun ishlatiladi.

Misol uchun:

```
int i=77;
float euler=2.718282;
void *vp;
vp=&I;
printf("%d", (*(int*)vp);
vp=&euler;
printf("%f", (*(float*)vp);
```

Quyidagi operatorlar ketma-ketligi xatolikka olib keladi:

```
void *vp; int *ip; ip=vp;
```


Bu xatolik sababi bitta obyektga har xil turdagi ko'rsatkichlar bilan murojaat qilish mumkin emas.

Ko'rsatkichlar funksiya parametri sifatida. Ko'rsatkichlar yordamida parametr qiymatini o'zgartirish mumkin.

Misol uchun, to'rtburchak yuzi va perimetrini berilgan tomonlari bo'yicha hisoblash funksiyasini quyidagicha tasvirlash mumkin.

```
void pr(float a, float b, float* s, float* p)
{
*p=2(a+b);
*s=a * b;
}
```

Bu funksiyaga quyidagicha murojaat qilinishi mumkin: pr(a, b, &p, &s). Funksiyaga p va s o'zgaruvchilarning adreslari uzatiladi. Funksiya tanasida *shu adreslar bo'yicha 2 * (a+b) va a * b qiymatlar yoziladi.*

Keyingi misolda berilgan ikki o'zgaruvchining qiymatlarini o'zaro almashtirish funksiyasidan foydalaniladi:

```
#include <stdio.h>
void change(int*a, int*b)//manzil bo'yicha uzatish
{
int r=*a; *a=*b; *b=r;
}
int main()
{
int x=1,y=5;
change(&x, &y);
printf("x=%d y=%d", x, y);
return 0;
}
```

Natija

x=5 y=1

Agar funksiya ichida parametrning o'zgarishini taqiqlash lozim bo'lib qolsa, bu holda const modifikatori qo'llanadi. Bu modifikatorni funksiyada o'zgarishi ko'zda tutilmagan barcha parametrlar oldidan qo'yish tavsiya qilinadi (undagi qaysi parametrlar o'zgaradi-yu, qaysilari o'zgarmasligi sarlavhadan ko'rinib turadi).

Ko'rsatkich va massiv nomi. Massivlar nomi dasturda konstanta *ko'rsatkichdir. Shuning uchun ham int z[4] massiv elementiga *(z+4)* shaklda murojaat qilish mumkin.

Massivlar bilan ishlanganda qavslarsiz ishlash mumkin.

Quyidagi misolda har bir harf alohida bosib chiqariladi:

```
#include <stdio.h>
int main()
{
char x[]="DIXI";
int i=0;
while (*(x+i)!='\0')
printf("\n %c", *(x+i++));
return 0;
}
```

Kompilator x[i] qiymatni hisoblaganda (x+i)*sizeof(< >) formula bo'yicha adresni hisoblab, shu adresdagi qiymatni oladi.

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

Massivlar funksiyalar parametrlari sifatida. Massivlar funksiyada bir o'lchamli massivlar sifatida yoki ko'rsatkichlar sifatida uzatilishi mumkin. Masalan, satrlar funksiyaga char turidagi bir o'lchamli massivlar *sifatida yoki char* turidagi ko'rsatkichlar sifatida uzatilishi mumkin.* Oddiy massivlardan farqli o'laroq, funksiyada satr uzunligi ko'rsatilmaydi, chunki satr oxirida satr oxiri '\0' belgisi bor.

Misol: Berilgan belgini satrda qidirish funksiyasi.

```
int find(char *s, char c)
{
for (int i=0; i<strlen(s); i++)
if(s[i] == c) return i;
return -1
}
```


Massiv funksiya qiymati sifatida. Massiv qiymati qaytaruvchi funksiya sifatida:

```
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 funksiyaga quyidagicha murojaat qilish mumkin:

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

Ko'p o'lchamli massivlar va ko'rsatkichlar. C da massivning eng umumiy tushunchasi – bu ko'rsatkichdir, bunda har xil turdagi ko'rsatkich bo'lishi mumkin, ya'ni massiv har qanday turdagi elementlarga, shu jumladan, massiv bo'lishi mumkin bo'lgan ko'rsatkichlarga ham ega bo'lishi mumkin. O'z tarkibida boshqa massivlarga ham ega bo'lgan massiv ko'p o'lchamli hisoblanadi.

Bunday massivlarni e'lon qilishda kompyuter xotirasida bir nechta turli xildagi obyekt yaratiladi. Masalan, int arr[4][3]

```
Arr
↓
arr[0]→ arr[0][0] arr[0][1] arr[0][2]
arr[1]→ arr[1][0] arr[1][1] arr[1][2]
arr[2]→ arr[2][0] arr[2][1] arr[2][2]
arr[3]→ arr[3][0] arr[3][1] arr[3][2]
```

Shunday qilib, arr[4][3] ning e'lon qilinishi dasturda uchta turli xildagi obyektlarni yuzaga keltiradi: arr identifikatorli ko'rsatkichni, to'rtta ko'rsatkichdan iborat nomsiz massivni va int turidagi o'n ikkita sondan iborat nomsiz massivni. Nomsiz massivlarga kirish huquqiga ega bo'lish uchun arr ko'rsatkichli adresli ifodalar qo'llaniladi. Ko'rsatkichlar massivi *elementlariga kirish huquqi arr[2] yoki *(arr+2) shaklidagi indeksli* ifodaning bittasini ko'rsatish orqali amalga oshiriladi. Butun int turidagi ikki o'lchamli sonlar massiviga kirish uchun arr[1][2] shaklidagi ikkita indeksli ifoda yoki unga ekvivalent bo'lgan *(*(arr+1)+2) va (*(arr+1))[2] shaklidagi ifodalar qo'llanilishi kerak.

5.4. Dinamik massivlar

Ko'rsatkichlar massivlari. Ko'rsatkichlar massivlari quyidagicha ta'riflanadi:

<tur> *<nom>[<son>]

Misol uchun int *pt[6] ta'rif int turidagi obyektlarga olti elementli massivni kiritadi.

Ko'rsatkichlar massivlari satrlar massivlarini tasvirlash uchun qulay.

Misol uchun familiyalar ro'yxatini kiritish uchun ikki o'lchovli massivdan foydalanish kerak.

char fam[][20]={"Olimov", "Raximov", "Ergashev"}

Xotirada 60 ta elementdan iborat bo'ladi, chunki har bir familiya 20 gacha 0 lar bilan to'ldiriladi.

Ko'rsatkichlar massivi yordamida bu massivni quyidagicha ta'riflash mumkin.

char *pf[]={"Olimov", "Raximov", "Ergashev"}.

Bu holda ro'yxat xotirada 23 ta elementdan iborat bo'ladi, chunki har bir familiya oxiriga 0 belgisi qo'yiladi

Har xil chegarali jadvallar bilan funksiyalardan foydalanishning bir yo'li – bu oldindan kiritiluvchi konstantalardan foydalanishdir. Lekin asosiy yo'li ko'rsatkichlar massivlaridan foydalanish.

Quyidagi misolda ko'rsatkichlar massivi yordamida so'zlar massivi tartiblanadi:

```
#include <stdio.h>
void sort(int n, char* a[])
{
char* c;
int i, j;
for (i=0; i<n; i++)
for (j=i+1; j<n; j++)
if (a[i][0]<a[j][0])
{
c=a[i]; a[i]=a[j]; a[j]=c;
};
};
int main()
{
char* pa[10]={"Alimov", "Dadashev", "Boboev"};
```



```
sort(3, pa);
for(int i=0; i<3; i++) printf("\n %s", pa[i]);
return 0;
}
```

Bu misolda shunga e'tibor berish kerakki, satrli massivning tartiblanmagan elementlarini almashtirish uchun qo'shimcha sikl kiritilgan edi. Ko'rsatkichlar massivining elementlari adreslardan iborat bo'lgani uchun, qo'shimcha siklga hojat qolmadi.

Xotira turlari. C tilida o'zgaruvchilar yo statik tarzda – kompilatsiya paytida, yoki standart kutubxonadan funksiyalarni chaqirib olish yo'li bilan dinamik tarzda – dasturni bajarish paytida joylashtirilishi mumkin. Asosiy farq ushbu usullarni qo'llashda ularning samaradorligi va moslashuvchanligi ko'rinadi. Statik joylashtirish samaraliroq, chunki bunda xotirani ajratish dastur bajarilishidan oldin sodir bo'ladi. Biroq bu usulning moslashuvchanligi ancha past, chunki bunda biz joylashtirilayotgan obyektning turi va o'lchamlarini avvaldan bilishimiz kerak bo'ladi. Masalan, matnli faylning ichidagi satrlarning statik massivida joylashtirish qiyin: avvaldan uning o'lchamlarini bilish kerak bo'ladi. Noma'lum sonli elementlarni oldindan saqlash va ishlov berish kerak bo'lgan masalalar odatda xotiraning dinamik ajratilishini talab qiladi.

Xotirani dinamik va statik ajratish o'rtasidagi asosiy farqlar quyidagicha:

- statik obyektlar nomlangan o'zgaruvchilar bilan belgilanadi hamda ushbu obyektlar o'rtasidagi amallar to'g'ridan to'g'ri, ularning nomlaridan foydalangan holda amalga oshiriladi. Dinamik obyektlar o'z shaxsiy otlariga ega bo'lmaydi va ular ustidagi amallar bilvosita, ko'rsatkichlar yordamida amalga oshiriladi;
- statik obyektlar uchun xotirani ajratish va bo'shatish kompilator tomonidan avtomatik tarzda amalga oshiriladi. Dasturchi bu haqida o'zi qayg'urishi kerak emas. Statik obyektlar uchun xotirani ajratish va bo'shatish to'laligicha dasturchi zimmasiga yuklatiladi. Bu anchayin qiyin masala va uni yechishda xatoga yo'l qo'yish oson.

Bir o'lchovli dinamik massivlar. Ma'lum bir turdagi elementlardan tashkil topgan berilgan o'lchamlardagi massivga xotira ajratish uchun **malloc** funksiyasidan foydalanish lozim. Afsuski, malloc funksiyasi massiv elementlarini initsiallashtirish imkonini bermaydi.

Masalan:

```
int *pia=(int*)malloc(4*sizeof(int));
```

Bu misolda xotira int turidagi to'rtta elementdan iborat massivga xotira ajratiladi.

Dinamik massivni bo'shatish uchun free funksiyasidan foydalanish lozim:

```
free(pia);
```

Misol:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
int*p=(int*)malloc(4*sizeof(int));
int i;
for(i=0; i<4; i++) p[i]=i;
for(i=0; i<4; i++) printf("%d", p[i]);
free(p);
return 0;
}
```

Natija:

```
0123
```

Ikki o'lchovli dinamik massivlar. Matritsani shakllantirishda oldin bir o'lchovli massivlarga ko'rsatuvchi ko'rsatkichlar massivi uchun xotira ajratiladi, keyin esa parametrli siklda bir o'lchovli massivlarga xotira ajratiladi.

Misol:

```
int n=3;
double **p=(double**)malloc(n*sizeof(double));
for (int i=0; i<n; i++)
p[i]=(double*)malloc(n*sizeof(double));
```

Xotirani bo'shatish uchun bir o'lchovli massivlarni bo'shatuvchi siklni bajarish zarur.

```
for(i=0; i<n; i++) free(p[i]);
free(p);
```

Quyidagi misolda dinamik matritsa hosil qilish ko'rsatilgan:


```
#include <stdio.h>
#include<stdlib.h>
int main()
{
int i, j, n=3;
double **p=(double**)malloc(n*sizeof(double));
for (int i=0; i<n; i++)
p[i]=(double*)malloc(n*sizeof(double));
for(i=0; i<n; i++)
for (j=0; j<n; j++)
p[j][i]=i * j+1;
for (i=0; i<n; i++){
printf("\n");
for (j=0; j<n; j++)
printf("%f", p[j][i]);
}
for(i=0; i<n; i++) free(p[i]);
free(p);
return 0;
}
```

5.5. Funksiyaga ko'rsatkich

Funksiyalarni chaqirishda foydalanish. C tili sintaksisiga ko'ra funksiyaga ko'rsatkich funksiya adresini aks ettiruvchi o'zgaruvchi yoki ifodadir. Funksiyaga ko'rsatkich bajariluvchi qiymati funksiya kodining birinchi bayti adresidir. Funksiyaga ko'rsatkichlar ustida arifmetik amallar bajarish mumkin emas. Eng keng qo'llanuvchi funksiyaga konstanta ko'rsatkich funksiyaning nomidir. Funksiyaga o'zgaruvchi ko'rsatkich funksiya ta'rifi va prototipidan alohida kiritiladi. Funksiyaga o'zgaruvchi ko'rsatkich quyidagicha ta'riflanadi:

<funksiya turi> (* ko'rsatkich nomi)(parametrlar spetsifikatsiyasi).

Misol uchun, int (*point) (void).

Bu ta'rifda qavslar muhim ahamiyatga ega, chunki qavslar yozilmasa, bu ta'rif parametrsiz funksiya prototipi deb qaraladi. Funksiyaga o'zgaruvchi ko'rsatkich qiymatlari sifatida, bir xil turga ega bo'lgan har xil funksiyalar adreslari berilishi mumkin.

Qiymati biror funksiya adresiga teng bo'lgan funksiyaga o'zgaruvchi ko'rsatkich shu funksiyaga murojaat qilish uchun ishlatilishi mumkin.

Quyidagi misolda funksiyaga uch xil murojaat qilish ko'rsatilgan:

```
#include<stdio.h>
void f1()
{
printf("\nf1 funksiya bajarildi");
};
void f2()
{
printf("\nf2 funksiya bajarilmadi");
};

int main()
{
void (*ptr) (void);
ptr=f1;
(*ptr)();
ptr=f2;
(*ptr)();
return 0;
}
```

Dasturda funksiyaga konstanta ko'rsatkich, ya'ni nomlari orqali va o'zgaruvchi ko'rsatkichlar yordamida murojaat qilishning hamma usullari ko'rsatilgan. Shuni ta'kidlash lozimki, adres olish * amali qo'llanilganda qavslar ishlatish shart.

Funksiyaga o'zgaruvchi ko'rsatkich ta'riflanganda instializatsiya qilish, ya'ni boshlang'ich qiymat sifatida o'zgaruvchi ko'rsatkich bilan bir xil turga ega bo'lgan funksiya adresini ko'rsatish mumkin. Misol uchun:

int fic (char);
int (*pfic) (char)=fic;

Funksiyaga ko'rsatkichlar massivlari. C tilida funksiyalar massivlarni yaratish mumkin emas, lekin funksiyaga ko'rsatkichlar massivlarini kiritish mumkin. Bunday massivlar quyidagicha ta'riflanadi:

<tur>(*massiv_nomi[hajm])(parametrlar spetsifikatsiyasi).

Tur – funksiyalar qaytaradigan qiymatlar turlari;

Massiv – nomi ixtiyoriy identifikator;

Hajm – massiv elementlari soni;

Parametrlar spetsifikatsiyasi – funksiyalar parametrlari nomlari va turlarini aniqlaydi.

Misol uchun int (*parray [4]) (char);

Bu massiv elementlari qiymatlari quyidagi prototiplarga ega bo'lgan funksiyalar adreslaridir: int funksiya_nomi (char).

Funksiyaga ko'rsatkichlar massivlari initsializatsiya qilinishi mumkin.

Misol uchun:

int f1(void);
int f2(void);
int (*pf[])()={f1, f2};

Bu ta'rifni typedef ta'riflovchisi yordamida soddalashtirish mumkin:

typedef int(*array_fync)(void);
array_func pf[]={f1, f2};

Indeks aniq qiymati bo'yicha massiv elementlariga va aniq funksiya-larga murojaat quyidagi shaklda amalga oshiriladi:

Massiv_nomi[indeks] (haqiqiy parametrlar_ro'yxati);
(* Massiv_nomi[indeks]) (haqiqiy parametrlar_ro'yxati);

Funksiyaga ko'rsatkichlar massivlari menyular yaratishda qulay. Quyida shunday menyu misolini ko'ramiz:

```
#include <stdio.h>
#include <stdlib.h>
#define n 2
void act0(char* name)
{
printf("%s: Ish tugadi!\n", name);
}
void act1(char* name)
{
printf("%s: Ish 1\n", name);
}
void act2(char* name)
{
printf("%s: Ish 2\n", name);
}
int main()
{
void (*pact[])(char*)={act0, act1, act2};
char string[12]="Bajarildi";
int number;
```

```
printf("\n Ish nomerini kiriting 0 dan %d gacha", n);
while(1)
{
scanf("%d", &number);
pact[number](string);
if (number == 0) break;
}
return 0;
}
```

Funksiyaga ko'rsatkichlar parametr sifatida. Funksiyaga ko'rsatkichlarni funksiyalarga parametr sifatida uzatish mumkin.

Bunday uzatish nomi oldindan belgilanmagan funksiyalar bilan ishlashga imkon beradi. Misol uchun to'rtburchaklar usuli yordamida integral hisoblash funksiyasidan foydalanilgan dasturni qarab chiqamiz. Dasturda $x/(x^2+1)^2$ funksiyasi integrali – 1 va 2 oraliqda hisoblanadi.

```
#include<stdio.h>
double ratio(double x)
{
double z;
z=x * x+1;
return x / (z * z);
};
double rectangle(double (*pf)(double), double a, double b)
{
int n=20;
int i;
double h, s=0.0;
h=(b – a) / n;
for (i=0; i<n; i++)
s+ =pf(a+h/2+i * h);
return h * s;
};
int main()
{
double a, b, c;
a=–1;
```



```
b=2.0;
c=rectangle(ratio, a, b);
printf("%f", c);
return 0;
}
```

Bu dasturda integralni hisoblash rectangle funksiyasini chaqirish orqali bajariladi.

Bu funksiya quyidagi parametrlarga ega: pf – double turli parametrga ega va shu turdagi qiymat qaytaruvchi funksiyaga ko'rsatkich, a va b parametrlari integrallash chegaralaridir. Integrallanuvchi funksiya qiymatlari ratio() funksiyani chaqirish orqali hisoblanadi. Dasturda rectangle() funksiyasi chaqirilib, pf ko'rsatkich qiymati ratio() funksiyasi adresiga teng.

Dastur bajarilishi natijasi:

0.149847

Funksiyaga ko'rsatkich funksiya qiymati sifatida. Menyular tashkil qilganda funksiyaga ko'rsatkichlarni qiymat sifatida qaytaruvchi funksiyalardan foydalanish qulay. Shu usulda menyu tashkil qilishni ko'rsatuvchi dasturni ko'rib chiqamiz. Dasturda uchta funksiya kiritilgan: int f(void) prototipiga ega bo'lgan f1() va f2() funksiyalari va int (*menu(void)) (void) prototipiga ega bo'lgan menyu() funksiyasi. Menyu() funksiyasi bajarilganda f1() va f2() funksiyalarga mos keluvchi menyu punktlaridan birini tanlash imkoni beriladi. Shu punktlardan biri tanlanganda mos funksiya adresi qaytariladi, agar punktlar noto'g'ri tanlangan bo'lsa, NULL qiymati qaytariladi. Bu qiymatlar asosiy dasturda r ko'rsatkichga uzatiladi.

Agar qiymat NULL bo'lsa, "The End" ma'lumoti chiqarilib, dastur bajarilishi to'xtatiladi, aks holda t=(*r)() murojaat adresi qaytarilgan funksiya bajarilishiga olib keladi.

```
#include <stdio.h>
int f1(void)
{
printf("The first actions:");
return 1;
}
int f2(void)
{
```

```
printf("The second actions:");
return 2;
}
int (*menu(void))(void)
{
int choice;
int (*menu_items[])()={f1, f2};
printf("\n Pick the menu item (1 or 2):");
scanf("%d", &choice);
if (choice<3&&choice>0)
return menu_items[choice-1];
else
return NULL;
}
int main()
{int (*r)(void);
int t;
while(1)
{r=menu();
if (r==NULL)
{
printf("\nThe End!");
return;
}
t=(*r)();
printf("|tt=%d", t);
}
return 0;
}
```

5.6. Strukturaga ko'rsatkichlar

Strukturaga ko'rsatkich ta'rifi. Strukturaga ko'rsatkichlar oddiy ko'rsatkichlar kabi ta'riflanadi:

Complex *cc, *ss; goods *p_goods;

Strukturaga ko'rsatkich ta'riflanganda initsializatsiya qilinishi mumkin. Misol uchun ekrandagi rangli nuqtani tasvirlovchi quyidagi strukturali

tur va strukturalar massivi kiritiladi. Strukturaga ko'rsatkich qiymatlari initsializatsiya va qiymat berish orqali aniqlanadi:

Struct point
{int color;
int x, y;
} a, b;
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, quyidagi shaklda qo'llaniladi:

(*strukturaga ko'rsatkich).element nomi;

Ikkinchi usul maxsus strelka (->) amaliga asoslangan bo'lib, quyidagi ko'rinishga ega:

strukturaga ko'rsatkich->element nomi

Struktura elementlariga quyidagi murojaatlar o'zaro tengdir:

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

Struktura elementlari qiymatlarini ko'rsatkichlar yordamida quyidagicha o'zgartirish mumkin:

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

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

struct particle * pinta=&m_point;

Bu holda m_point struktura elementlarini quyidagicha o'zgartirish *mumkin:*

Pinta->mass=18.4;
for (i=0; i<3; i++)
Pinta->coord[i]=0.1 * i;

Strukturaga ko'rsatkich parametr sifatida. Funksiyada struktura qiymatini o'zgartirish uchun adres orqali uzatish lozim.

```
#include<stdio.h>
struct point
{
int x, y;
```

```
}
void print_point(struct point a)
{
printf("%d %d", x, y);
}
void input_point(struct point *pa)
{
scanf ("%d %d", &pa->x, &pa->y);
}
int main()
{
struct point a;
input_point(&a);
print_point(a);
return 0;
}
```

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 bo'yicha uzluksiz siljish mumkin bo'ladi.

Misol tariqasida kompleks sonlar massivi summasini hisoblash masalasini ko'rib chiqamiz:

```
#include <stdio.h>
struct complex
{
float x;
float y;
};
int main()
{
struct complex 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++;
}
printf("\n Summa: real = %f", summa.x);
printf("imag = %f", summa.y);
return 0;
}
```

Dastur bajarilishi natijasi:
Summa: real = –8.000000, imag = –16.000000.

5-bob bo'yicha savollar

1. Ko'rsatkich ta'rifini keltiring.
2. Ilova ko'rsatkichdan qanday farq qiladi?
3. Ko'rsatkichlar bilan bog'liq amallarni keltiring.
4. Ko'rsatkich va massiv nomi orasida qanday farq bor?
5. Ko'rsatkichlar massivi qanday ta'riflanadi?
6. Dinamik xotira ajratish va bo'shatish funksiyalari.
7. Dinamik massivlar oddiy massivlardan qanday farq qiladi?
8. Dinamik massivlarni hosil qilish usullarini ko'rsating.
9. Funksiyaga ko'rsatkich ta'rifining umumiy ko'rinishi.
10. Funksiyaga ko'rsatkich massivlari.

5 bob bo'yicha topshiriqlar

1. Berilgan satr o'zgaruvchi ekanligini aniqlovchi funksiya tuzing va dasturda foydalaning. Funksiya tanasida faqat ko'rsatkichlar ustida amallardan foydalaning.
2. Matritsani vektorga ko'paytirish funksiyasini yaratib dasturda foydalaning. Matritsa ko'rsatkichlar massivi sifatida kiritilsin.
3. Dinamik ravishda o'quvchilar familiyalari va baholari massivlarini hosil qiluvchi funksiyalar yarating. Hamma a'lochilar familiyalarini chiqaruvchi funksiya tuzib dasturda foydalaning.
4. Uchburchak dinamik massiv yordamida Paskal uchburchagini hisoblovchi funksiya tuzing. Bu funksiyadan tashqari uchburchakni ekranga chiqaruvchi funksiya tuzib, dasturda foydalaning.
5. Dixotomiya usuli yordamida f(x) = 0 tenglamani yechish uchun funksiya tuzing. Funksiyaga ko'rsatkich parametr sifatida uzatilsin.

6-bob. FAYLLAR BILAN ISHLASH

6.1. Fayllar

C tilining asosiy xususiyatlaridan biri oldindan rejalashtirilgan fayllar strukturasi yo'qligidir. Hamma fayllar baytlar ketma-ketligi deb ko'riladi. UNIX operatsion tizimda har bir qurilmaga «Maxsus fayl» mos keladi, shuning uchun C kutubxonasidagi funksiyalar fayllar bilan ham, qurilmalar bilan ham ma'lumot almashinishi uchun foydalaniladi. C tili kutubxonasida kiritish-chiqarish, quyi darajadagi kiritish, chiqarish va portlar uchun kiritish-chiqarish, oqimli daraja tizim xususiyatlariga bog'liq bo'lishi uchun bu yerda qaralmaydi.

Oqimli kiritish va chiqarish. Oqimli kiritish va chiqarishda ma'lumotlar bilan almashish baytma-bayt amalga oshiriladi. Lekin tashqi xotira qurilmalari bilan almashish oldidan belgilangan ma'lumotlar bloki orqali amalga oshiriladi. Odatda, bu blokning minimal hajmi 512 yoki 1024 baytga teng bo'ladi. Diskdan, ya'ni fayldan o'qishda ma'lumotlar operatsion tizim buferiga yoziladi, so'ngra baytma-bayt yoki ma'lum porsiyalar bilan foydalanuvchi dasturiga uzatiladi. Diskka, ya'ni faylga yozishda buferga yig'iladi, so'ngra diskka murojaat qilinganda yagona blok sifatida uzatiladi. Buferlar operativ xotira qismlari sifatida yaratiladi, shuning uchun ma'lumot almashishi diskka to'g'ridan to'g'ri murojaat qilishidan ko'ra tezroq amalga oshadi. Shunday qilib, oqim bu buferlash vositalari va fayldir.

Oqim bilan ishlashda quyidagi vazifalarni bajarish mumkin:

- oqimlarni ochish va yopish;
- simvol, qator, satr, formatlangan ma'lumot ixtiyoriy uzunlikdagi ma'lumotlarni kiritish yoki chiqarish va fayl oxiriga yetganlik shartini tahlil qilish;
- buferlash va bufer hajmini boshqarish;
- ko'rsatkichni oqimdagi o'rnini aniqlash yoki yangi o'ringa ko'chirish.

Bu vazifalarni bajaruvchi funksiyalardan foydalanish uchun dasturga stdio.h – faylini ulash lozim.

Dastur bajarilishi boshlanganda avtomatik ravishda quyidagi oqimlar ochiladi:

- standart kiritish oqimi stdin;
- standart chiqarish oqimi stdout;
- xatolar haqida ma'lumotlar standart oqimi stderr.

Oqimlarni ochish va yopish. Oqim ochilishi uchun oldindan kiritilgan FILE turidagi struktura bilan bog'lash lozimdir. FILE strukturasi ta'rifi stdio.h kutubxonasida joylashgan.

Bu strukturada buferga ko'rsatkich, o'qilayotgan pozitsiyaga ko'rsatkich va boshqa ma'lumotlar saqlanadi.

Oqim ochilganda dasturga oqimga ko'rsatkich, ya'ni FILE strukturali turdagi obyektga ko'rsatkich qaytariladi. Bu ko'rsatkich quyidagicha e'lon qilinishi lozim.

FILE * <ko'rsatkich nomi>

Misol uchun FILE * fp

Oqim ochish funksiyasi quyidagi ko'rinishga ega:

<oqimga ko'rsatkich nomi> = fopen(<fayl-nomi>, <ochish rejimi>)

Misol uchun:fp=fopen("t.txt", "r")

Oqim bilan bog'liq faylni quyidagi rejimlarda ochish mumkin:

"w" – yangi fayl o'qish uchun ochiladi. Agar fayl mavjud bo'lmasa, qaytadan yaratiladi;

"r" – mavjud fayl faqat o'qish uchun ochiladi;

"a" – fayl davom ettirish uchun ochiladi;

"w+" – fayl yozish va keyingi tahrirlash uchun ochiladi. Fayl ixtiyoriy joyidan o'qish yoki yozish mumkin;

"r+" – fayl ixtiyoriy joyidan o'qish yoki yozish mumkin, lekin fayl oxiriga qo'shish mumkin emas;

"a+" – fayl ixtiyoriy joyidan o'qish va yozish uchun ochiladi. Quyidagi "w+" rejimdan farqli fayl oxiriga ma'lumot qo'shish mumkin.

Matnli rejimda oqimdan o'qilgan quyidagi simvollar CR (qiymati 13) «karetkani qaytarish» va LF (qiymati 10) – «yangi qator boshiga o'tish» bitta simvolga "\n" (qiymati LF ya'ni 10ga teng) simvolga almashtiradi.

Agar fayl matnli emas, ixtiyoriy ma'lumotni saqlasa, binar rejimda ochiladi. Buning uchun rejimlar belgilariga b harfi qo'shiladi, masalan,

"wb" yoki "r+b". Ba'zi kompilatorlarda matnli rejim t harfi yordamida ko'rsatiladi, masalan, «rt».

Oqim ochilganda quyidagi xatolar kelib chiqishi mumkin: ko'rsatilgan fayl mavjud emas (o'qish rejimida); disk to'la yoki yozishdan himoyalangan va hokazo. Yana shuni aytish kerakki, fopen( ) funksiyasi bajarilganda dinamik xotira ishlatiladi. Agar xotirada joy qolmagan bo'lsa, «not enough memory» xatosi kelib chiqadi.

Ko'rsatilgan hollarda ko'rsatkich NULL qiymatga ega bo'ladi.

Bu xatolar haqidagi ma'lumotlarni ekranga chiqarish uchun perror () funksiyasi ishlatiladi. Bu funksiya stdio.h kutubxonasida saqlanuvchi prototipi quyidagi ko'rinishga ega.:

void perror(const char * s);

Diskda ochilgan fayllarni berkitish uchun quyidagi funksiyadan foydalaniladi.

int fclose (<oqimga-ko'rsatkich nomi>).

6.2. Faylga ketma-ket murojaat qilish

Fayllar bilan ishlashning bitli rejimi. Fayl bilan bitli almashish rejimi getc() va putc() funksiyalari yordamida tashkil etiladi. Bu funksiyalarga quyidagi shaklda murojaat etiladi:

c=getc(fp);

putc(c, fp);

Bu yerda fp–ko'rsatkich

C – int turidagi o'zgaruvchi

Misol tariqasida klaviaturadan simvol kiritib faylga yozishni ko'ramiz. Matn oxirini '#' belgisi ko'rsatadi. Fayl nomi foydalanuvchidan so'raladi. Agar <enter> klavishi bosilsa, faylga CR va LF (qiymatlari 13 va 10) konstantalar yoziladi. Keyinchalik fayldan simvollarni o'qishda bu konstantalar satrlarni ajratishga imkon beradi.

```
#include <stdio.h>
void main()
{
FILE *fp;
char c;
const char CR='\015';
const char LF='\012';
```



```
char fname[20];
puts("fayl nomini kiriting:\n");
gets(fname);
if((fp=fopen(fname, "w"))==NULL)
{
perror(fname);
return 1;
}
while ((c=getchar())!='#')
{
if (c=='\n')
{putc(CR, fp);
putc(LF, fp);
}
else putc (c, fp);
}
fclose(fp);
}
```

Keyingi dastur fayldan simvollarni o'qib, ekranga chiqaradi:

```
#include <stdio.h>
#include<conio.h>
int main()
{
FILE *fp;
char c;
char fname[20];
puts("fayl nomini kiriting:\n");
gets(fname);
if((fp=fopen(fname, "r"))==NULL)
{
perror(fname);
return 1;
}
while ((c=getc(fp))!=EOF)
putchar(c);
fclose (fp);
```

```
getch();
}
```

Satrlar yordamida fayllar bilan bog'lanish. Matnli fayllar bilan ishlash uchun fget va fputs funksiyalaridan foydalaniladi. Bu funksiyalar prototiplari stdio.h faylida quyidagi ko'rinishga ega:

int fputs (const char *s, FILE *stream);

char *fgets (char * s, int n, FILE * stream);

fputs funksiyasi '\0' simvoli bilan chegaralangan satrni stream ko'rsatkichi orqali aniqlangan faylga yozadi. '\0' simvoli faylga yozilmaydi.

fgets() funksiyasi stream ko'rsatkichi orqali aniqlangan fayldan (n–1) simvolni o'qiydi va s ko'rsatgan satrga yozib qo'yadi. Funksiya n–1 simvolni o'qib bo'lsa yoki 1-qator simvoli '\n'ni uchratsa, ishini to'xtatadi. Har bir satr oxiriga qo'shimcha '\0' belgisi qo'shiladi. Xato bo'lganda yoki fayl oxiriga yetganda agar fayldan birorta simvol o'qilmagan bo'lsa, NULL qiymat qaytariladi.

Misol tariqasida fayl nomini foydalanuvchidan so'rab yaratuvchi va bu faylga ikkita kiritilgan so'zni yozib qo'yuvchi dasturni ko'rib chiqamiz:

```
#include <stdio.h>
#include <conio.h>
void main()
{
FILE *fp;
char s[256];
char fname[20];
int n;
puts("fayl nomini kiriting:\n");
gets(fname);
if((fp=fopen(fname, "w"))==NULL)
{
perror(fname);
getch();
return;
}
for(n=1; n<3; n++) {
gets(s);
```



```
fputs(s, fp);
fputs("\n", fp);
}
fclose(fp);
getch();
}
```

Keyingi misolda nomi kiritilgan fayldan monitorga o'qishni ko'ramiz:

```
#include <stdio.h>
#include <conio.h>
void main()
{
FILE *fp;
char s[256];
char fname[20];
int n;
puts("fayl nomini kiriting:\n");
gets(fname);
if((fp=fopen(fname, "r"))==NULL)
{
perror(fname);
getch();
return;
}
for(n=1; n<3; n++) {
fgets(s, 256, fp);
puts(s);
}
fclose(fp);
getch();
}
```

Quyidagi dasturda bir fayldagi matnni ikkinchi faylga yozishni ko'rib chiqamiz.

```
#include <stdio.h>
#include <conio.h>
```

```
void main()
{
FILE *f1, *f2;
char s[256];
char fname1[20];
char fname2[20];
puts("fayl nomini kiriting:\n");
gets(fname1);
if((f1 = fopen(fname1, "r")) = = NULL)
{
perror(fname1);
getch();
return;
}
puts("fayl nomini kiriting:\n");
gets(fname2);
if((f1 = fopen(fname2, "w")) = = NULL)
{
perror(fname2);
getch();
return;
}
while (fgets(s, 256, f1)! = NULL)
fputs(s, f2);
fclose(f1);
fclose(f2);
getch();
}
```

Fayllar bilan formatli almashinuv. Ko'p hollarda ma'lumotni to'g'-ridan to'g'ri monitorga chiqarishda qulay shaklda faylda saqlash zarur bo'ladi. Bu holda faylga formatli kiritish va chiqarish funksiyalaridan foydalanish mumkin. Bu funksiyalar quyidagi prototiplarga ega:

int fprintf(oqimga ko'rsatkich, formatlash qatori, o'zgaruvchilar ro'y-xati);

int fscanf (oqimga ko'rsatkich, formatlash qatori, o'zgaruvchilar ro'y-xati);

Misol tariqasida fayl nomi foydalanuvchidan so'rab yaratuvchi va bu faylga 1 dan 100 gacha bo'lgan sonlarning simvolli tasvirini yozib qo'yuvchi dasturni ko'rib chiqamiz:

```
#include <stdio.h>
void main()
{
FILE *fp;
int n;
char fname[20];
puts("fayl nomini kiriting:\n");
gets(fname);
if((fp = fopen(fname, "w")) = = NULL)
{
perror(fname);
return;
}
for(n = 1; n < 11; n++)
fprintf(fp, "%d", n);
fclose(fp);
}
```

Keyingi misolda nomi kiritilgan fayldan monitorga o'qishni ko'ramiz:

```
#include <stdio.h>
#include <conio.h>
void main()
{
FILE *fp;
int n;
char fname[20];
puts("fayl nomini kiriting:\n");
gets(fname);
if((fp = fopen(fname, "r")) = = NULL)
{
perror(fname);
getch();
return;
}
for(n = 1; n < 11; n++) {
```

```
fscanf(fp, "%d", &n);
printf("%d", n);
}
fclose(fp);
getch();
}
```

Standar oqimga chiqarish. Keyingi dasturda satrlarni standart kiritish oqimi, ya'ni klaviaturadan kiritish va standart chiqarish oqimiga, monitorga chiqarish ko'rsatilgan:

```
#include<stdio.h>
#define MAXLINE 20
int main(void)
{
char line[MAXLINE];
while (fgets(line, MAXLINE, stdin) !=NULL &&
line[0] !='\n')
fputs(line, stdout);
return 0;
}
```

6.3. Faylga ixtiyoriy murojaat qilish

Ixityoriy kiritish va chiqarish. Hozirgi ko'rib chiqilgan funksiyalar faylga ketma-ket yozish yoki ketma-ket o'qishga imkon beradi xolos. Fayldan o'qib faylga yozishlar doim joriy pozitsiyada bo'ladi. Boshlang'ich pozitsiya fayl ochilganda aniqlanadi. Faylni "r" va "w" rejimida ochilganda joriy pozitsiya ko'rsatkichi faylning birligi baytini ko'rsatadi, "a" rejimida ochilganda, oshish baytini ko'rsatadi. Har bir kiritish-chiqarish amali bajarilganda, ko'rsatkich o'qilgan baytlar soniga qarab yangi pozisiyaga ko'chadi. Faylning ixtiyoriy baytiga murojaat qilish uchun fseek () funksiyasidan foydalanish lozim. Bu funksiya quyidagi prototipga ega:

int fseek (faylga ko'rsatkich, oraliq, hisobot boshi) farq log turidagi o'zgaruvchi yoki ifoda bilan beriladi. Hisobot boshi oldin quyidagi konstantalardan biri bilan aniqlanadi:

SEEK_ SET(qiymati 0) – fayl boshi;

SEEK_CUR (qiymati 1) – o'qilayotgan pozitsiya;

SEEK_ END (qiymati 2) – fayl ochish.

fseek () funksiyasi 0 qaytaradi, agar faylda ko'chish bajarilgan bo'lsa, aksincha, noldan farqli songa teng bo'ladi.

Ixtiyoriy pozitsiyadan fayl boshiga o'tish:

fseek (fp, ol, SEEK_SET)

Ixtiyoriy 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).

Quyidagi misolda fayldan simvoollar avval to'g'ri tartibda, so'ngra teskari tartibda o'qiladi:

```
#include <stdio.h>
#include <conio.h>
int main(void)
{
char file[256];
char ch;
FILE *fp;
long count, last=0;
puts("fayl nomini kiriting:\n");
gets(file);
if ((fp=fopen(file, "r+"))== NULL)
{
printf("faylni ochib bo'lmadi %s\n", file);
exit(1);
}
while(!feof(fp))
{
ch=getc(fp);
putchar(ch);
putchar('\n');
last++;
}
for (count=1L; count<=last; count++)
```

```
{
fseek(fp, -count, SEEK_END);
ch=getc(fp);
putchar(ch);
putchar('\n');
}
fclose(fp);
getch();
}
```

fseek funksiyasidan tashqari C tili kutubxonasida pozitsiyaga ko'rsatkichlar bilan bog'liq quyidagi funksiyalar mavjud.

long ftell (FILE*) – faylda ko'rsatkichning joriy pozitsiyasini aniqlash.

void rewind (FILE*) – joriy pozitsiya ko'rsatkichini fayl boshiga keltirish.

Quyidagi misolda nomi kiritilgan fayl ikkilik rejimda faqat o'qish uchun ochiladi. So'ngra fayl ko'rsatkichi fayl oxiriga keltirilib, ftell funksiyasi yordamida fayl hajmi aniqlanadi. So'ngra rewind funksiyasi yordamida fayl ko'rsatkichi fayl boshiga qaytarilib, siklda simvollar ketma-ket fayldan o'qilib, ekranga chiqariladi.

```
#include <stdio.h>
#include <conio.h>
void main()
{
char file[256];
char ch;
FILE *fp;
long count, last;
puts("fayl nomini kiriting:\n");
gets(file);
if ((fp=fopen(file, "rb"))== NULL)
{
printf("faylni ochib bo'lmadi %s\n", file);
exit(1);
}
fseek(fp, 0L, SEEK_END);
last=ftell(fp);
```



```
rewind(fp);
for (count=1L; count<=last; count++)
{
ch=getc(fp);
putchar(ch);
putchar('\n');
}
fclose(fp);
getch();
}
```

Keyingi misolda nomi kiritilgan fayl simvollari oxiridan boshiga qarab o'qiladi:

```
#include <stdio.h>
#include <conio.h>
int main(void)
{
char file[256];
char ch;
FILE *fp;
long count, last;
puts("fayl nomini kiriting:\n");
gets(file);
if ((fp=fopen(file, "rb"))== NULL)
{
printf("faylni ochib bo'lmadi %s\n", file);
exit(1);
}
fseek(fp, 0L, SEEK_END);
last=ftell(fp);
for (count=1L; count<=last; count++)
{
fseek(fp, -count, SEEK_END);
ch=getc(fp);
putchar(ch);
putchar('\n');
}
```

```
fclose(fp);
getch();
}
```

Murakkab ma'lumotlarni o'qish va yozish. Murakkab ma'lumotlarni kiritish va chiqarish fread() va fwrite() funksiyalari orqali amalga oshiriladi. Bu funksiyalar prototiplari quyidagi ko'rinishga ega:

size_t fread(void * ptr, size_t size, size_t nmemb, FILE * fp);

size_t fwrite(const void * ptr, size_t size, size_t nmemb, FILE *fp);

Ikkala funksiya butun o'qilgan yoki yozilgan baytlar sonini qaytaradi. fread funksiyasi fp fayl ko'rsatkichi bilan ochilgan fayldan nmemb sonli size parametrida ko'rsatilgan miqdordagi baytlarni o'qib, ptr ko'rsatkichi orqali ko'rsatilgan buferga yozadi. O'qish fayldagi joriy pozitsiyadan boshlanadi.

Masalan, fayldan 10 ta double turidagi sonni o'qib, massivga yozish:

```
double earnings[10];
fread(earnings, sizeof (double), 10, fp);
```

fwrite() funksiyasi fp fayl ko'rsatkichi bilan ochilgan faylga nmemb sonli size parametrida ko'rsatilgan miqdordagi baytlarni, ptr ko'rsatkichi orqali ko'rsatilgan buferdan yozadi. Yozish fayldagi joriy pozitsiyadan boshlanadi.

Masalan, faylga 10 ta double turidagi sonni massivdan yozish:

```
double earnings[10];
fwrite(earnings, sizeof (double), 10, fp);
```

Faylga 256 bayt ma'lumot yozish:

```
char buffer[256];
fwrite(buffer, 256, 1, fp);
```

Quyida struktura turidagi massivni faylga yozish ko'rsatilgan:

```
#include <stdio.h>
#include <conio.h>
typedef struct
{
char name[64];
int age;
int salary;
} employee;

int main(void)
```



```
{int i;
char file[256];
FILE* fp;
employee ww[3]={{"AA", 1,1}, {"BB", 2,2}, {"CC", 3,3.0}};
puts("fayl nomini kiriting:\n");
gets(file);
if ((fp=fopen(file, "wb"))== NULL)
{
printf("faylni ochib bo'lmadi %s\n", file);
exit(1);
}
fwrite(&ww[0], sizeof(employee), 3, fp);
fclose(fp);
getch();
}
```

Quyida struktura turidagi massivni fayldan o'qish ko'rsatilgan:

```
#include <stdio.h>
#include <conio.h>
typedef struct
{
char name[64];
int age;
int salary;
} employee;

int main(void)
{int i;
char file[256];
FILE* fp;
employee ww[3];
puts("fayl nomini kiriting:\n");
gets(file);
if ((fp=fopen(file, "rb"))== NULL)
{
printf("faylni ochib bo'lmadi %s\n", file);
exit(1);
}
```

```
fread(&ww[0], sizeof(employee), 3, fp);
for(i=0; i<3; i++)
printf("\n %s %d %f", ww[i].name, ww[i].age, ww[i].salary);
fclose(fp);
getch();
}
```

6.4. Quyi darajadagi kiritish va chiqarish

Quyi darajadagi kiritish va chiqarish funsiyalari operatsion tizim imkoniyatlaridan to'g'ridan to'g'ri foydalanishga imkon beradi. Bu holda buferlash va formatlash bajarilmaydi. Faylni quyi darajadagi ochishda fayl bilan fayl (oqim) ko'rsatkichi emas, deskriptor bog'lanadi. Fayl deskriptori fayl ochilganligi to'g'risidagi ma'lumotni operatsion tizim ichki jadvallariga joylashtiruvini belgilovchi butun sondir. Quyi darajadagi funksiyalar dasturga stdio.h kutubxonasini qo'shishni talab qilmaydi. Lekin bu kutubxona fayllar bilan ishlashda foydali bo'lgan ba'zi konstantalar (misol uchun, fayl yakuni belgisi EOF) ta'rifini o'z ichiga oladi. Bu konstantalardan foydalanganda stdio.h dasturga qo'shilishi zarur.

Fayllarni ochish va yopish. Fayllarni quyi darajada ochish uchun open () funksiyasidan foydalaniladi:

int fd=open (fayl nomi, bayroqlar, murojaat.)

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 konstantalardan razryadli '|' amali yordamida hosil qilingan bo'lishi mumkin.

Konstantalar ro'yxati:

O_APPEND	faylni oxiriga 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 funksiyasi xatolik 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 matnli rejimda ochish;
O_TRUNK	mavjud faylni ochish va bor ma'lumotni o'chirish.

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

S_IWRITE	faylga yozishga ruxsat berish;
S_IREAD	fayldan o'qishga ruxsat berish;
S_IREAD\S_WRITE	o'qish va yozishga ruxsat berish.

Ko'rsatilgan konstantalar sys katalogida joylashgan stat.h sarlavhali 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 ruxsat beriladi. UNIX operatsion tizimida murojaat huquqlari 3 xil foydalanuvchilar uchun ko'rsatiladi:

fayl egasi;

foydalanuvchilar guruhi a'zosi;

boshqa foydalanuvchilar.

Foydalanuvchilar huquqlari quyidagi simvollar orqali ko'rsatiladi:

R – fayldan o'qish ruxsat berilgan;

W – faylga yozish ruxsat berilgan;

X – fayllarni bajarish ruxsat berilgan.

Agar biror murojaat huquqi berilmagan bo'lsa, o'rniga '–' 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 o'rniga 0 raqami, 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 () funksiyasida murojaat huquqi parametri sifatida ko'rsatiladi. Faylni ochishga misollar:

1. Faylni o'qish uchun ochish:

fd=open ("t.txt", O_RDONLY)

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)

Faylni ochishda kelib chiqadigan xato turini aniqlash uchun errno.h sarlavhali faylda saqlanuvchi errno o'zgaruvchisi xizmat qiladi. Agar bu o'zgaruvchi qiymati shu sarlavhali faylda saqlanuvchi EEXIST konstantasiga teng bo'lsa, ochilayotgan fayl mavjudligini bildiradi.

sopen() funksiyasi bitta faylga bir necha dasturlardan murojaat qilish imkonini beradi. Albatta, dasturlar faylga faqat o'qish rejimida murojaat qilishi mumkin. Faylni ochish uchun yana creat () funksiyasi mavjud bo'lib, quyidagi open () funksiyasini chaqirishga mos keladi.

open(fayl nomi, O_CREAT |O_TRUNK| O_WRONLY); bu funksiya yangi fayl yaratadi va yozish uchun ochadi. Quyi darajada fayllarni yopish uchun close () funksiyasidan foydalanish lozim. Bu funksiya ko'rinishi quyidagicha:

int close (fayl deskriptori). Funksiya muvaffaqiyatli bajarilganda 0 qaytaradi. Xato bo'lganda – 1.

Ma'lumotlarni o'qish va yozish. Quyi darajada ma'lumotlarni kiritish va chiqarish read() va write() funksiyalari orqali amalga oshiriladi. Bu funksiyalar prototiplari quyidagi ko'rinishga ega:

int read (int fd, char * buffer; unrigned int count)

int write (int fd, char * buffer; unsiqned int count)

Ikkala funksiya butun o'qilgan yoki yozilgan baytlar sonini qaytaradi. read funksiyasi fd deskriptori bilan ochilgan fayldan count parametrida ko'rsatilgan miqdordagi baytlarni o'qib, buffer ko'rsatkichi orqali ko'rsatilgan bufferga yozadi. Fayl oxiriga yetganda read () funksyasi 0 qiymat qaytaradi. Fayldan o'qishda xatolik kelib chiqsa, –1 qiymat qaytaradi. O'qish fayldagi joriy pozitsiyadan boshlanadi. Agar fayl matnli rejimda ochilsa, CR va LF simvollari '\n' simvoliga o'zgartiriladi.

Yozish write() funksiyasi 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' simvoli CR va LF simvollar sifatida yoziladi. Agar yozishda xatolik kelib chiqsa, write () funksiyasi –1 qiymat qaytaradi. Xatoni aniqlash errno global o'zgaruvchisi bo'lsa, errno.h sarlavhali faylda ko'rsatilgan quyidagi konstantalar birga teng bo'ladi.

EACCES – fayl yozuvdan himoyalangan;

ENOSPC – tashqi qurilmada bo'sh joy qolmagan;

EBADF – noto'g'ri fayl deskriptori.

Bu funksiyalar io.h sarlavhali faylda joylashgan. Quyida bir fayldan ikkinchisiga nusxa olish dasturini ko'rib chiqamiz:

```
#include <stdio.h>
#include <fcntl.h>
#include <io.h>
int main(int argc, char *argv[ ])
{
int fdin, fdout; /*Fayllar deskriptorlari */
int n; /* O'qilgan baytlar soni */
char buff[BUFSIZ];
if (argc !=3)
{
printf ("Dastur chaqirish formati:");
printf ("\n %s birinchi_fayl ikkinchi_fayl",
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;
```

```
}
/* fayllar ochilgan */
while ((n=read(fdin, buff, BUFSIZ))>0)
write (fdout, buff, n);
return 0;
} /* dastur oxiri */
```

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

Faylga ixtiyoriy murojaat. Quyi darajada fayllarni ixtiyoriy tartibda o'qish mumkin. Buning uchun lseek () funksiyasidan foydalanish lozim. Bu funksiya prototipi quyidagi ko'rinishga ega:

long lseek (int fd, long offset, int origin);

Bu funksiya fd deskriptori bilan bog'liq fayldagi joriy pozitsiyani uchinchi parametr (origen) orqali nuqtaga nisbatan ikkinchi parametr (offset) qadamga ko'taradi. Boshlang'ich 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 oxiri.

Ko'chish davomida xato kelib chiqsa, xato kodi errno global o'zgaruvchisiga yoziladi. Faylda joriy pozitsiyani aniqlash uchun tell () funksiyasidan foydalaniladi:

Bu funksiya prototipi: long tell (int fd);

Joriy pozitsiyani fayl boshiga keltirish:

lseek (fd, oh, SEEK_SET)

Joriy pozitsiyani fayl oxiriga keltirish:

lseek (fd, oh, SEEK_END)

Misol:

```
#include <sys\stat.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>
int main(void)
```



```
{
int handle;
char msg[]="This is a test";
char ch;

/* create a file */
handle=open("TEST.SSS", O_CREAT | O_RDWR, S_IREAD |
S_IWRITE);

/* write some data to the file */
write(handle, msg, strlen(msg));
/* seek to the begining of the file */
lseek(handle, 0L, SEEK_SET);

/* reads chars from the file until we hit EOF */
do
{
read(handle, &ch, 1);
printf("%c", ch);
} while (!eof(handle));
close(handle);
return 0;
}
```

6-bob bo'yicha savollar

1. Oqim deb nimaga aytiladi?
2. Oqim ochish funksiyasining tuzilishi.
3. Oqim ochish rejimlari.
4. Simvolli kiritish va chiqarish.
5. Satrli kiritish va chiqarish.
6. Formatli kiritish va chiqarish.
7. Ixtiyoriy tartibda kiritish va chiqarish qanday amalga oshiriladi?
8. Murakkab ma'lumotlarni kiritish va chiqarish uchun qanday funksiyalardan foydalanadi?
9. Quyi darajada fayllarni ochish usullarini ko'rsating.
10. Qaysi funksiyalar quyi darajada o'qishga va yozishga imkon beradi?

1. Abiturient (ismi, tug'ilgan yili, yig'ilgan ball, attestat o'rta bali) strukturasini yarating. Struktura turidagi massiv yarating va faylga yozing. Ko'rsatilgan raqamli elementni olib tashlang va ko'rsatilgan familiyali elementdan so'ng element qo'shing.
2. Xodim (ismi, lavozimi, tug'ilgan yili, oylik maoshi) strukturasini yarating. Struktura turidagi massiv yarating va faylga yozing. Ko'rsatilgan familiyali elementni olib tashlang va ko'rsatilgan raqamli elementdan so'ng element qo'shing.
3. Mamlakat (nomi, poytaxti, aholi soni, egallagan maydoni) strukturasini yarating. Struktura turidagi massiv yarating va faylga yozing. Ko'rsatilgan aholi sonidan kichik bo'lgan elementni olib tashlang va ko'rsatilgan nomga ega bo'lgan elementdan so'ng element qo'shing.
4. Davlat (nomi, davlat tili, pul birligi, valuta kursi) strukturasini yarating. Struktura turidagi massiv yarating va faylga kiriting. Ko'rsatilgan nomga ega bo'lgan elementni o'chiring va fayl oxiriga ikkita element qo'shing.
5. Inson (ismi, yashash manzili, telefon raqami, yoshi) strukturasini yarating. Struktura turidagi massiv yarating va faylga yozing. Ko'rsatilgan yoshga ega bo'lgan elementni o'chiring va berilgan telefon raqamidagi elementdan oldin element qo'shing.

7-bob. PREPROTSESSOR VOSITALARI

7.1. Preprotsessor tushunchasi

Dastur matni va preprotsessor. C tilida matnli fayl shaklida tayyorlangan dastur uchta qayta ishlash bosqichlaridan o'tadi.

Matnning preprotsessor direktivalari asosida o'zgartilishi. Bu jarayon natijasi yana matnli fayl bo'lib, preprotsessor tomonidan bajariladi.

Kompilatsiya. Bu jarayon natijasi mashina kodiga o'tkazilgan obyektli fayl bo'lib, kompilator tomonidan bajariladi.

Bog'lash. Bu jarayon natijasi to'la mashina kodiga o'tkazilgan bajariluvchi fayl bo'lib, bog'lang'ich (komponovshik) tomonidan bajariladi.

Preprotsessorning vazifasi dastur matnini preprotsessor direktivalari asosida o'zgartirishdir. Masalan, define direktivasi dasturda bir jumlani ikkinchi jumla bilan almashtirish uchun ishlatiladi. Bu direktivaning umumiy ko'rinishi quyidagicha:

#define <almashtiruvchi ifoda> <almashinuvchi ifoda>

Bu direktiva bajarilganda dastur matnidagi almashtiruvchi ifodalar almashinuvchi ifodalarga almashtiriladi.

Dasturda quyidagi matn mavjud bo'lsin:

```
#define EULER 2.718282
double mix=EULER
d=alfa*EULER
```

Preprotsessor 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
```

Matndagi almashtirishlarni #undef direktivasi orqali rad etish mumkin.

Masalan:

```
#define E 2
```

```
#undefine E
#define E 'A'
```

include direktivasi ikki ko'rinishda ishlatilishi mumkin.

#include *fayl nomi* direktivasi dasturning shu direktiva o'rniga qaysi matnli fayllarni qo'shish kerakligini ko'rsatadi.

#include <*fayl nomi*> direktivasi dasturga kompilator standart kutubxonalariga mos keluvchi sarlavhali fayllar matnlarini qo'shish uchun mo'ljallangan. Bu fayllarda funksiya prototipi, turlar, o'zgaruvchilar, konstantalar ta'riflari yozilgan bo'ladi. Funksiya prototipi funksiya qaytaruvchi tur, funksiya nomi va funksiyaga uzatiluvchi turlardan iborat bo'ladi. Misol uchun cos funksiyasi prototipi quyidagicha yozilishi mumkin: double cos(double). Agar funksiya nomidan oldin void turi ko'rsatilgan bo'lsa, bu funksiya hech qanday qiymat qaytarmasligini ko'rsatadi. Shuni ta'kidlash lozimki, bu direktiva dasturga standart kutubxona qo'shilishiga olib kelmaydi. Standart funksiyalarning kodlari bog'lash, ya'ni aloqalarni tahrirlash bosqichida, kompilatsiya bosqichidan so'ng amalga oshiriladi.

Kompilatsiya bosqichida sintaksis xatolar tekshiriladi va dasturda bunday xatolar mavjud bo'lmasa, standart funksiyalar kodlarisiz mashina kodiga o'tkaziladi.

Sarlavhali fayllarni dasturning ixtiyoriy joyida ulash mumkin bo'lsa ham, bu fayllar, odatda, dastur boshida qo'shish lozim. Shuning uchun bu fayllarga sarlavhali fayl (header file) nomi berilgan.

Dastur matnini kompilatsiya qilish. Dastur kodini bajariluvchi faylga o'tkazish uchun kompilatorlar qo'llaniladi. Kompilator qanday chaqiriladi va unga dastur kodi joylashgan joyi haqida qanday xabar qilinadi, bu konkret kompilatorga bog'liq. Bu ma'lumotlar kompilatorning hujjatlarida berilgan bo'ladi.

Dastur kodi kompilatsiya qilinishi natijasida obyektli fayl hosil qilinadi. Bu fayl, odatda, obj kengaytmali bo'ladi. Lekin bu hali bajariluvchi fayl degani emas. Obyektli faylni bajariluvchi faylga o'girish uchun yig'uvchi dastur qo'llaniladi.

Yig'uvchi dastur yordamida bajariluvchi faylni hosil qilish. C tilida dasturlar, odatda, bir yoki bir nechta obyektli fayllar yoki kutubxonalarni komponovka qilish yordamida hosil qilinadi. Kutubxona deb bir yoki bir nechta komponovka qilinuvchi fayllar to'plamiga aytiladi. C ning barcha

kompilatorlari dasturga qo'shish mumkin bo'lgan funksiyalar (yoki protseduralar) va sinflardan iborat kutubxona hosil qila oladi. Funksiya – bu ayrim xizmatchi amallarni, masalan, ikki sonni qo'shib, natijasini ekranga chiqarishni bajaruvchi dastur blokidir. Sinf sifatida ma'lumotlar to'plami va ularga bog'langan funksiyalarni qarash mumkin. Funksiyalar va sinflar haqidagi ma'lumotlar keyingi mavzularda batafsil berilgan.

Demak, bajariluvchi faylni hosil qilish uchun quyida keltirilgan amallarni bajarish lozim:

Avval .cpp kengaytmali dastur kodi hosil qilinadi;

Dastur kodini kompilatsiya qilish orqali .obj kengaytmali obyektli fayl tuziladi;

Bajariluvchi faylni hosil qilish maqsadida .obj kengaytmali fayli zaruriy kutubxonalar orqali komponovka qilinadi.

Fayllardan matnlar qo'shish. Fayldan matn qo'shish 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 qo'llanilsa, preprotsessor qo'shilayotgan faylni standart kutubxonalardan izlaydi.

Agar ikkinchi shakl qo'llanilsa, preprotsessor foydalanuvchining joriy katalogini ko'rib chiqadi va bu katalogda fayl topilmasa, standart tizimli kataloglarga murojaat qiladi.

Agar dasturda bir necha funksiyalardan foydalanilsa, funksiyalar ta'rifi, tanasi bilan birga alohida fayllarda saqlanishi qulay. Hamma funksiyalar tanasiga va main() funksiyasi tanasiga chaqirilayotgan funksiyalar prototiplari joylashtirilsa, dastur tanasida funksiyalarni ixtiyoriy joylashtirish mumkin. Bu holda dastur faqat protsessor komandalaridan ham iborat bo'lishi mumkin.

C standarti bo'yicha .h suffiksi kutubxonaga tegishli funksiyalarning prototiplari hamda turlar va konstantalar ta'rifi joylashgan fayllarni ko'rsatadi. Bunday fayllar sarlavhali fayllar deb ataladi. Kompilator kutubxonalari bilan ishlashga mo'ljallangan sarlavhali fayllar ro'yxati til standartida ko'rsatilgan bo'lib, bu fayllar nomlari tilning xizmatchi so'zlari hisoblanadi.

Quyida shu standart fayllar nomlari keltirilgan:

assert.h – dastur diagnostikasi .
type.h – simvollarni o'zgartirish va tekshirish.
erruo – xatolarni tekshirish.
float.h – haqiqiy sonlar bilan ishlash.
limits.h – butun sonlarning chegaralari.
locate.h – milliy muhitga moslash.
math.h – matematik hisoblashlar.
setjump.h – nolokal o'tishlar imkoniyatlari.
signal.h – g'ayrioddiy holatlar bilan ishlash.
stdarg.h – o'zgaruvchi sonli parametrlarni qo'llash.
stddef.h – qo'shimcha ta'riflar.
stdlib.h – xotira bilan ishlash.
string.h – simvolli qatorlar bilan ishlash.
time.h – sana va vaqtni aniqlash.

Turbo C va Borland C++ kompilatorlarida grafik kutubxona bilan bog'lanish uchun graphic.h – sarlavhali fayl qo'llaniladi.

Shartli direktivalar. Shartli direktiva quyidagi ko'rinishga ega:

#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 kompilatsiya 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 mo'ljallangan fayllarning har biriga bitta fayl ulanish mo'ljallangan bo'lsa, bu fayl bir necha marta dasturga ulanib qoladi. Bu qayta ulanishning oldini olish uchun standart fayllar yuqorida ko'rilgan

direktivalar yordamida himoya qilingan. Bu himoya usuli quyidagicha bo'lishi mumkin:

```
/* filename Nomli fayl */
/* FILENAME aniqlanganligini tekshirish */
# indef FILE_NAME
... /* Ulanayotgan fayl ...
Ta'rif */
#define FILE_NAME
#endif
```

Tarmoqlanuvchi shartli direktivalar yaratish uchun quyidagi direktiva kiritilgan:

#elif *butun_sonli_ifoda*

Bu direktiva ishlatilgan matn strukturasi:

```
#if shart
matn
#elif 1_ifoda
1_matn
#elif 2_ifoda
2_matn
...
#else
matn
#endif
```

Preprotsessor avval #if direktivasidagi shartni tekshiradi. Agar shart 0 ga teng bo'lsa, 1_ifoda hisoblanadi, agar u ham 0 bo'lsa, 2_ifodani hisoblaydi va hokazo.

Agar hamma ifodalar 0 bo'lsa, else uchun ko'rsatilgan matn ulanadi. Agar biror ifoda 0 dan katta bo'lsa, shu direktivada ko'rsatilgan matn ulanadi.

Unar defined operatsiyasi. Matn shartli qayta ishlanganda unar preprotsessor amali defined operand amalidan foydalanish mumkin.

Quyidagi if defined ifodasi #ifdef operand ifodasiga ekvivalent. Bu ko'rinishda defined afzalligi bilinmaydi.

Misol uchun biror tekst kompilatorga Y identifikatori aniqlangan, N esa aniqlanmagan holda uzatish lozim bulsin. U holda preprotsessor direktivasi quyidagicha yoziladi:

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

Bu direktivani quyidagicha ham yozish mumkin:

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

Yordamchi direktivalar. Satrlarni nomerlash uchun quyidagi direktivadan foydalanish mumkin:

#line *konstanta*

Direktiva faqat satr nomeri emas, fayl nomini ham o'zgartirishi mumkin:

#line *konstanta "fayl nomi"*

Odatda, bu direktiva kam ishlatiladi.

Quyidagi direktiva leksemalar ketma-ketligi orqali ko'rsatilgan 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 tekshirib, 5 ga teng bo'lmagan holda ma'lumot berish uchun quyidagi direktivadan foydalaniladi:

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

Xech qanday xizmat bajarmaydigan direktiva:

#

7.2. Makroslar

Makros tushunchasi. Makros ta'rifiga ko'ra bir jumlani ikkinchi jumla bilan almashtirishdir. Makroslar ko'pincha makrota'rif deb ham ataladi. Eng sodda makrota'rif

define *identifikator almashtiruvchi satr.*

Bu direktiva yordamida foydalanuvchi asosiy turlar uchun yangi nomlar kiritishi mumkin.

Masalan: # define real long double

Dastur matnida long double turidagi o'zgaruvchilarni real sifatida ta'riflash mumkin.

Parametrli makrota'riflardan foydalanish yanada kengroq imkoniyatlar yaratadi:

define nom (parametrlar ro'yxati) almashtiriluvchi_qator

Bu yerda nom – makros nomi.

Parametrlar ro'yxati – vergul bilan ajratilgan identifikatorlar ro'yxati.

Makrota'rifning klassik misoli:

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

Bu makrosdan foydalanganda kompilator max (a, b) ifodani (x<a ? y:x) ifoda bilan almashtiradi.

Yana bir 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.

Makroslarning funksiyadan farqi. Funksiya dasturda bitta nusxada bo'lsa, makros hosil qiluvchi matnlar makros har gal chaqirilganda dasturga joylashtiriladi. Funksiya parametrlar spetsifikatsiyasida ko'rsatilgan turlar uchun ishlatiladi va konkret turdagi qiymat qaytaradi. Makros har qanday turdagi parametrlar bilan ishlaydi. Hosil qilinayotgan qiymat turi faqat parametrlar turlari va ifodalarga bog'liq.

Makrojoylashlardan to'g'ri foydalanish uchun almashtiriluvchi satrni qavsga olish foydali.

Funksiyaning haqiqiy parametrlari bu ifodalar, makros argumentlari bo'lsa, vergul bilan ajratilgan leksemalardir. Argumentlarga makro-kengaytirishlar qo'llanmaydi.

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) printf(#A"=%f", A)

makro ta'rif berilgan bo'lsin. U holda makrosga print(a+b) murojaat quyidagi makrokengaytmani hosil qiladi: print("a+b" "%f", a+b).

Ikkinchi '##' amal leksemalar orasida qo'llanilib, leksemalarni ulashga imkon beradi.

Quyidagi makro ta'rif berilgan bo'lsin:

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

Makrochaqiriq: Makrojoylash natijasi:

```
Zero(+, x, y) (bac)
One(+, x, y) (x+y)
Two(+, x, y) (x+y)
```

Birinchi holda bac yagona identifikator deb qaralib, makroalmashtirish amalga oshirilmaydi. Ikkinchi holda makros argumentlari bo'shliq belgilari bilan ajratilgan bo'lib, bu belgilar natijada ham saqlanib qoladi. Uchinchi holda makros uchun '##' amali qo'llanilgani uchun, natijada, bo'shliq belgilarsiz parametrlar ulanadi.

Oldindan kiritilgan makronomlar. Protsessorga qayta ishlash jarayonida ma'lumot beruvchi quyidagi oldindan kiritilgan makronomlar mavjud:

..LINE.. – qiymati o'nlik konstanta o'qilayotgan satr nomeri. Birinchi satr nomeri 1 ga teng.

..FILE.. – fayl nomi simvollar qatori sifatida. 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, kun, 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'rsatadi.

..STDC.. – Agar kompilator ANSI – standart bo'yicha ishlayotgan bo'lsa, qiymati 1 ga teng konstanta. Aks holda konstanta qiymati aniqlanmagan.

7.3. Xotira sinflari

Avtomatik xotira obyektlari. *Blok* deb funksiya tanasi yoki figurali qavslar ichiga olingan ta'riflar va operatorlar ketma-ketligiga aytiladi.

Avtomatik xotira faqat o'zi aniqlangan blok ichida mavjud bo'ladi. Blokdan chiqishda obyektlar uchun ajratilgan xotira qismi bo'shatiladi, ya'ni obyektlar yo'qoladi. Shunday qilib, avtomatik xotira har doim ichki xotiradir, ya'ni bu xotiraga o'zi aniqlangan blokda murojaat qilish mumkin. Avtomatik xotira obyektlari auto xizmatchi so'zi yordamida ta'riflanadi.

Ko'zda tutilgani bo'yicha har qanday lokal xotira obyekti avtomatik xotiraga tegishli bo'ladi.

Registrli xotira obyektlari. Registrli xotira obyektlari hamma jihatlari bo'yicha avtomatik xotira obyektlari bilan bir xil bo'ladi, faqat ular uchun xotira, iloji bo'lsa, registrlarda ajratiladi. Registrli xotira obyektlari register xizmatchi so'zi yordamida ta'riflanadi.

Statik xotira obyektlari. *Statik xotira* obyektlari blokdan chiqilgandan so'ng ham mavjud bo'lib qolaveradi. Statik xotira obyektlari static xizmatchi so'zi yordamida ta'riflanadi.

Misol:

```
#include <stdio.h>
void autofunc(void)
{
int K=1;
printf("K=%d", K);
K++;
return;
}
int main()
{
int i;
for (i=0; i<5; i++)
autofunc();
return 0;
}
```

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 <stdio.h>
void autofunc(void)
{
static int K=1;
printf("K=%d", K);
K++;
return;
```

```
}
int main()
{
int i;
for (i=0; i<5; i++)
autofunc();
return 0;
}
```

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.

Global obyektlar. *Global* obyektlar deb dasturda hamma bloklar uchun umumiy bo'lgan obyektlarga aytiladi. Har bir blok ichida global obyektga murojaat qilish mumkin.

```
#include <stdio.h>
int N=5;
void func(void)
{
printf("\tN=%d", N);
N--;
return;
}
int main()
{
for (int i=0; i<5; i++)
{
func();
N+=2;
}
return 0;
}
```

Dastur bajarilishi natijasi:

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

Bunda N o'zgaruvchisi main() va func() funksiyalari tashqarisida aniqlangan va bu funksiyalarga nisbatan globaldir. Har bir func() chaqirilganda uning qiymati 1 ga kamayadi, asosiy dasturda esa 2 taga oshadi. Natijada N qiymati 1 ga oshib boradi.

Endi dasturni o'zgartiramiz:

```
#include <stdio.h>
int N=5;
void func(void)
{
printf("\tN=%d", N);
N--;
return;
}
int main()
{
int N;
for (int i=0; i<5; i++)
{
func();
}
return 0;
}
```

Dastur bajarilishi natijasi:

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

N o'zgaruvchisi main() funksiyasida avtomatik o'zgaruvchi sifatida ta'riflangan va u global N o'zgaruvchiga ta'sir qilmaydi. Ya'ni global N o'zgaruvchi main() funksiyada ko'rinmaydi.

Tashqi obyektlar. Dastur bir necha matnli fayllarda joylashgan bo'lishi mumkin. Dastur o'z navbatida funksiyalardan iborat. Hamma funksiyalar tashqi hisoblanadi. Hatto har xil fayllarda joylashgan funksiyalar ham har xil nomlarga ega bo'lishi lozim. Funksiyalardan tashqari dasturlarda tashqi obyektlar, o'zgaruvchilar, ko'rsatkichlar va massivlar ishlatilishi mumkin.

Tashqi obyektlar hamma funksiyalarda ham ko'rinmasligi mumkin. Agar obyekt fayl boshida ta'riflangan bo'lsa, u shu fayldagi hamma funksiyalarda ko'rinadi.

Agar tashqi obyektga shu obyekt ta'riflangan blokdan yuqorida yoki boshqa faylda joylashgan blokdan murojaat qilinishi kerak bo'lsa, bu obyekt extern xizmatchi so'zi yordamida ta'riflanishi lozim. Shuni aytish lozimki, extern xizmatchi so'zi yordamida ta'riflanganda initsializatsiya qilish yoki chegaralarni ko'rsatish 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 funksiyalari boshqa faylda ta'riflangan bo'lsin. Bu holda bu fayllar orasidagi bog'liqlikni ta'minlash uchun quyidagi ta'riflar lozim:

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() ...
```

Dinamik xotira. Dinamik xotira – bu dastur bajarilishi jarayonida ajratiladigan xotiradir. Dinamik xotira malloc funksiyasi orqali ajratilgandan so'ng, to free funksiyasi tomonidan bo'shatilmaguncha saqlanadi.

Quyidagi misolda yakka obyektga xotira ajratiladi:

```
int*pint=(int*)malloc(sizeof(int))
```

Bu yerda malloc funksiyasi int turidagi nomsiz obyektga xotirani ajratib beradi hamda yaratilgan obyekt adresini qaytarib beradi. Bu adres pint ko'rsatkichiga joylashtiriladi. Ushbu nomsiz obyekt ustidagi barcha hatti-harakatlar shu ko'rsatkich bilan ishlash orqali amalga oshiriladi, chunki dinamik obyekt bilan to'g'ridan to'g'ri ish olib borish (manipulatsiyalar o'tkazish) mumkin emas.

Agar dinamik xotira dastur bajarilishi tugaguncha bo'shatilmagan bo'lsa, avtomatik ravishda dastur tugaganda bo'shatiladi. Shunga qaramay ajratilgan xotirani dasturda maxsus bo'shatish dasturlashning sifatini oshiradi.

Dinamik obyekt kerak bo'lmay qolganda, unga ajratilgan xotirani to'g'ridan to'g'ri bo'shatish free funksiyasi yordamida amalga oshiriladi:

```
free(pint);
```

Dastur bajarilishi davomida ajratilgan xotira qismiga murojaat imkoniyati shu qismga adreslovchi ko'rsatkichga bog'liq. Shunday qilib, biror blokda ajratilayotgan dinamik xotira bilan ishlashning quyidagi uchta varianti mavjud:

- ko'rsatkich avtomatik xotira turiga kiruvchi lokal obyekt. Bu holda ajratiladigan xotiraga blokdan tashqarida murojaat qilib bo'lmaydi, shuning uchun blokdan chiqishda bu xotirani bo'shatish lozim;
- ko'rsatkich avtomatik xotira turiga kiruvchi lokal statik obyekt. Blokda bir marta ajratilgan dinamik xotiraga, blokka har bir qayta kirilganda ko'rsatkich orqali murojaat qilish mumkin. Xotirani blokdan foydalanib bo'lgandan so'ng bo'shatish lozim;
- ko'rsatkich blokka nisbatan global obyekt. Dinamik xotiraga ko'rsatkich ko'rinuvchi hamma bloklardan murojaat qilish mumkin. Xotirani faqat undan foydalanib bo'lgandan so'ng bo'shatish lozim.

Ikkinchi variantga misol keltiramiz, bu misolda dinamik xotira obyekti ichki statik ko'rsatkich bilan bog'liq:

```
#include <stdio.h>
#include<stdlib.h>
void dynamo(void)
{
static char *uc=NULL;
if (uc==NULL)
{
uc=(char*)malloc(1);
*uc='A';
}
printf("%c", *uc);
(*uc)++;
return;
};
int main()
{
int i;
```

```
for (i=0; i<5; i++)
dynamo();
return 0;
}
```

Dastur bajarilishi natijasi:

A B C D E

Bu dasturning kamchiligi ajratilgan xotira bo'shatilmasligidir.

Keyingi dasturda dinamik xotiraga ko'rsatkich global obyektdir:

```
#include <stdio.h>
#include<stdlib.h>
char *uk=NULL;
void dynam1(void)
{
printf("%c", *uk);
(*uk)++;
return;
};
int main()
{
int i;
uk=(char*)malloc(1);
*uk='A';
for (i=0; i<5; i++)
{
dynam1();
(*uk)++;
}
free(uk);
return 0;
}
```

Dastur bajarilishi natijasi:

A C E G I

Dinamik obyekt asosiy dasturda yaratilib, uk ko'rsatkich bilan bog'liq. Dasturda boshlang'ich 'A' qiymatga ega bo'ladi. Ko'rsatkich global bo'lgani uchun dinamik obyektga main() va dynam1() funksiyalarida murojaat qilish mumkin.

Dinamik xotiraga ajratilgandan so'ng shu obyekt bilan bog'liq ko'rsatkich tashqi obyekt sifatida ta'riflangan ixtiyoriy blokda murojaat qilish mumkin.

7.4. Bosh funksiya parametrlari

Main funksiyasi parametrlari. Har qanday dastur quyidagicha sarlavhaga ega bo'lishi lozim:

int main (int argc, char*argv[ ], char*envp[ ])

argv – satrlarga ko'rsatkichlar massivi;

argc – int turidagi parametr argv massividagi elementlar sonini belgilaydi;

envp – har biri muhit o'zgaruvchilaridan birini ta'riflovchi satrlarga ko'rsatkichlar massivi.

Muhit deyilganda main() funksiyasini ishga tushirgan operatsion tizim tushuniladi.

Asosiy main() funksiyasi parametrlari vazifasi bajarilayotgan dastur bilan operatsion tizim, aniqrog'i, dasturni ishga tushirgan komanda qatori bilan aloqani ta'minlashdan iborat.

Agar main() funksiyasi ichida funksiyani ishga tushirgan komanda qatoridagi ma'lumotga ehtiyoj bo'lmasa, parametrlar tashlab ketiladi.

argv [ ] massivi indeksi 0 dan boshlanadi. Bu element dasturning nomi bilan birga to'la yo'lni ko'rsatadi.

Misol uchun dastur nomi Example bo'lib, u komanda qatoridagi quyidagi satr orqali ishga tushirilayotgan bo'lsin.

C:\ CATALOG\ Example.exe

Bu holda argc qiymati 1 ga teng bo'ladi, Argv[0] esa quyidagi satrni ko'rsatadi:

"C:\ CATALOG\ Example.exe"

Misol uchun komanda qatoridan hamma ma'lumotni ekranga so'zma-so'z chiqaruvchi dasturni ko'ramiz:

```
#include <stdio.h>
int main(int argc, char* argv[ ])
{
int i;
for (i=0; i<argc; i++)
printf("\n %s", argv[i]);
```

```
return 0;
}
```

Dastur quyidagi komanda qatori orqali ishga tushirilsin:

C:\VVP\ tert66.exe 11 22 33

Dastur bajarilishi natijasi:

Argv[0] – C:_VVP\ tert 66.exe

Argv[1] – 11;

Argv[2] – 22;

Argv[3] – 33;

Odatda, argv ko'rsatkichlar massivi NULL qiymat bilan tugaydi. Bu xossadan foydalanib, yuqoridagi massivni birinchi parametrsiz yozish mumkin:

```
#include <stdio.h>
int main(int argc, char* argv[ ])
{
int i;
for (i=0; argv[i]!=NULL; i++)
printf("\n %s", argv[i]);
return 0;
}
```

Uchinchi parametr envp vazifasi dasturga muhit haqidagi ma'lumotni uzatish uchun ishlatiladi. Bu parametrning imkoniyatlarini quyidagi dasturda ko'ramiz.

```
#include <stdio.h>
int main(int argc, char* argv[ ], char*envp[ ])
{
int n;
printf("\n dastur parametrlar soni%d", argc-1);
for (n=0; n<argc; n++)
printf("\n%dchi parametr:%s", n, argv[n]);
for (n=0; envp[n]; n++)
printf("\n%s", envp[n]);
return 0;
}
```


7.5. Parametrlar soni o'zgaruvchi bo'lgan funksiyalar

Ixtiyoriy sonli parametrlar. C tilida parametrlar soni belgilanmagan funksiyalar yaratish mumkin. Parametrlar soni va turi funksiyaga murojaat qilinganda, haqiqiy parametrlar soni va turi asosida aniqlanadi. Parametrlar soni o'zgaruvchan bo'lgan funksiya juda bo'lmasa bitta parametrga ega bo'lishi kerak.

Parametrlar soni o'zgaruvchan funksiya ta'rifi:

<tur> <ism>(<oshkor parametrlar>, ...)

Verguldan so'ng uch nuqta keyingi parametrlar soni va turi ixtiyoriy bo'lishi mumkinligini ko'rsatadi. Ro'yxat boshi va oxirini belgilash uchun ikki usul mavjud:

ro'yxat tugashi belgisi ma'lum;

uzatilayotgan parametrlar soni ma'lum.

```
#include <stdio.h>
int sum(int n, ...)
{
int total=0;
int arg;
int*pa=&n;
for(int i=0; i<n; i++)
{
pa++;
arg=*pa;
total +=arg;
};
return total;
};
int main()
{
int k=9;
int m=sum(4, 1, k, k, 1);
printf("%d", m);
return 0;
}
```

Ixtiyoriy sonli parametrlar uchun makrovositalar. Parametrlar soni o'zgaruvchi funksiyalar yaratishning qulay usuli stdarg.h faylida saqlanuvchi makrota'riflardan foydalanishdir. Bu makrota'riflar ixtiyoriy sonli parametrlarga ega bo'lgan funksiyalar yaratishning qulay va sodda vositalari bo'lib, quyidagi ko'rinishga ega:

void **va_start**(va_list param, oxirgi aniq parametr) – ro'yxatning birinchi elementi bilan bog'lanish;

<tur> **va_arg**(va_list param, <tur>) – ro'yxatning keyingi elementini o'qish;

void **va_end**(va_list param) – ro'yxat tugagandan so'ng chaqiriladigan komanda.

Bu makroslarda birinchi parametrlar stdarg.h faylida aniqlangan maxsus va_list turiga tegishli bo'lishi lozim. Bu tur orqali ko'rsatkich xossasiga ega bo'lgan obyekt e'lon qilinadi. Funksiya tanasida albatta va_list turidagi obyekt ta'riflangan bo'lishi lozim. Misol uchun

va_list factor;

Bu obyekt va_start() funksiyasi yordamida noma'lum uzunlikdagi ro'yxatning birinchi elementi bilan bog'lanadi. Buning uchun makrosning ikkinchi parametri sifatida oxirgi ta'riflangan parametr ishlatiladi:

va_start(factor, oxirgi_aniq parametr);

Bu funksiya factor ko'rsatkichiga oxirgi aniq parametr adresini qiymat sifatida beradi.

Noma'lum ro'yxat birinchi parametr turini funksiyaga uzatish va ko'rsatkichni shu parametrga to'g'rilash uchun quyidagi murojaatdan foydalanish lozim:

va_arg(factor, <tur>);

Keyingi parametr turini funksiyaga uzatish va ko'rsatkichni shu parametrga to'g'rilash uchun yana va_arg() makrosiga murojaat qilinadi:

va_arg(factor, <tur_1>);

Ixtiyoriy sonli parametrli funksiyadan to'g'ri qaytish uchun va_end makrosidan foydalaniladi:

va_end(factor);

Bu makros parametrlar ro'yxati tugaganda chaqiriladi. Shuni ko'rsatish lozimki va_start() makrosi chaqirilmasdan oldin va_end() makrosi qayta chaqirilishi mumkin emas.

```
#include <stdio.h>
#include <stdarg.h>
```



```
void miniprint(char *format, ...)
{
va_list ap;
char *p;
int ii;
double dd;
va_start(ap, format);
for (p=format; *p; p++)
{
if (*p!='%')
{
printf("%c", *p);
continue;
}
switch(*++p)
{
case 'd':ii=va_arg(ap, int);
printf("%d", ii);
break;
case 'f':dd=va_arg(ap, double);
printf("%d", dd);
break;
default:printf("%c", *p);
}
}
va_end(ap);
}

int main()
{
int k=154;
double e=2.718282;
miniprint("\n k=%d,\t e=%f", k, e);
return 0;
}
```

7-bob bo'yicha savollar

1. Kompilator va preprotsessorning farqi nimadan iborat?
2. Shartli direktivalar nima uchun ishlatiladi?
3. Xotira sinflarini ko'rsating.
4. Tashqi obyektlar qanday e'lon qilinadi?
5. Ta'rif bilan e'lon orasida qanday farq bor?
6. Dinamik xotira bilan ishlashning qanday variantlari mavjud?
7. Dinamik o'zgaruvchilar oddiy o'zgaruvchilardan qanday farq qiladi?
8. Bosh funksiya parametrlari.
9. Nima uchun parametrlari soni o'zgaruvchan funksiya juda bo'lmasa bitta parametrga ega bo'lishi kerak.
10. Parametrlar soni o'zgaruvchi funksiyalar yaratish uchun qanday makrota'riflardan foydalanish qulay?

7-bob bo'yicha topshiriqlar

1. Ikkita funksiya yaratib, ikki faylga yozing. Ikkala faylni dasturga ulab, funksiyalarni ishlating.
2. Ikki son minimumini hisoblovchi makros yarating va dasturda foydalaning.
3. Berilgan ketma-ketlik qat'iy kamayuvchi ekanligini tekshiruvchi parametrlar soni o'zgaruvchan funksiya tuzing va dasturda foydalaning.
4. Parametrlari soni o'zgaruvchan funksiya yordamida matritsani vektorga ko'paytirish funksiyasini yaratib, dasturda foydalaning. Matritsa ko'rsatkichlar massivi sifatida kiritilsin.
5. Hamma xotira sinflariga tegishli o'zgaruvchilar qatnashgan dastur tuzing.

8-bob. MATEMATIK VA SATRLI STANDART FUNKSIYALAR

8.1. Matematik funksiyalar

Quyida ko'riladigan matematik funksiyalardan foydalanish uchun dasturga <matn.n> sarlavhali fayl ulash lozim

Trigonometrik funksiyalar. Kosinus cos, sinus sin, tangens tan.

Prototiplari:

double cos(double x);
double sin(double x);
double tan(double x);

Argumentlar qiymatlari radianda beriladi.

Sinus va kosinus – 1 dan 1 gacha qiymat qaytaradi.

Tangens sin(x)/cos(x) formula bo'yicha hisoblanadi.

```
#include<stdio.h>
#include<math.h>
int main()
{
double x;
x=M_PI_2;
printf("x=%lf sin=%lf\n", x, sin(x));
x=0;
printf("x=%lf cos=%lf\n", x, cos(x));
x=M_PI_4;
printf("x=%lf cos=%lf\n", x, tan(x));
return 0;
}
```

Bu misolda M_PI_2 va M_PI_4 pi ning yarmi va to'rtdan bir qismini bildiruvchi konstantalardir.

Teskari trigonometrik funksiyalar. Teskari trigonometrik funksiyalar arkkosinus acos, arksinus asin, arktangens atan, ikki son x/y bo'linmasi arktangensi atan2 funksiyalari yordamida hisoblanadi.

double acos(double x);
double asin(double x);
double atan(double x);
double atan2(double y, double x);

Arkkosinus va arksinus argumentlari qiymati – 1 va 1 oralig'ida yotishi kerak.

Funksiyalar radianda qiymat qaytaradi

Arkkosinus qiymati 0 va pi oralig'ida.

Arksinus qiymati – pi/2 va pi/2 oralig'ida.

Arktangens qiymati – pi/2 va pi/2 oralig'ida

Bo'linma arktangensi qiymati– pi va pi oralig'ida.

Misol:

```
#include<stdio.h>
#include<math.h>
int main()
{
double x, y;
x=1;
printf("x=%lf asin=%lf\n", x, asin(x));
x=0;
printf("x=%lf acos=%lf\n", x, acos(x));
x=1500000;
printf("x=%lf atan=%lf\n", x, atan(x));
x=3000000; y=2;
printf("x=%lf y=%lf atan2=%lf\n", x, y, atan2(x, 1.0));
return 0;
}
```

Bu misolning bajarilishi natijasida hamma funksiyalar bitta qiymat, ya'ni pining yarmiga teng qiymat qaytaradi. Albatta atan va atan2 funksiyalari qiymatlari taxminan teng bo'ladi.

Absolut qiymat. Haqiqiy sonning absolut qiymati fabs funksiyasi yordamida hisoblanadi.

Prototipi:

double fabs(double x);

Misol:

```
#include<stdio.h>
#include<math.h>
int main()
{
double x=-2.0;
printf("x=%lf fabs=%lf\n", x, fabs(x));
return 0;
}
```

Logarifm. Natural logarifmni hisoblash uchun log, o'nlik logarifmni hisoblash uchun log10 funksiyalaridan foydalaniladi.

Prototiplari:

double log(double x);
double log10(double x);

Misol:

```
#include<stdio.h>
#include<math.h>
int main()
{
double x=2.718282;
printf("x=%lf log=%lf\n", x, log(x));
x=10.0;
printf("x=%lf log10=%lf\n", x, log10(x));
return 0;
}
```

Eksponenta. Eksponentani hisoblash uchun exp funksiyasidan foydalaniladi.

Prtotipi:

double exp(double x);

Misol:

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

```
double y;
double x=1.0;
y=exp(x);
printf("x=%lf exp(x)=%lf\n", x, y);
return 0;
}
```

Daraja. Darajani hisoblash uchun pow funksiyasidan foydalanish lozim.

Prototipi:

Double pow(double x, double y);

Funksiya x^y qiymatni qaytaradi.

Agar x va y ning ikkalasi 0 bo'lsa, natijasi 1.

Agar x<0 va y musbat bo'lsa, xatolik yuzaga keladi.

Misol:

```
#include<stdio.h>
#include<math.h>
int main()
{
double x=2.0, y=3.0;
printf("x=%lf y=%lf pow(x, y)=%lf\n", x, y, pow(x, y));
return 0;
}
```

Yaxlitlash funksiyalari. Sonni yuqoriga yaxlitlash uchun ceil va pastga yaxlitlash uchun floor funksiyasidan foydalaniladi.

Prototiplari:

double ceil(double x);

double floor(double x);

Birinchi funksiya x dan kichik bo'lmagan eng kichik butun sonni qaytaradi.

Ikkinchi funksiya x dan katta bo'lmagan eng katta butun sonni qaytaradi.

Ikkala funksiya butun sonni double turda qaytaradi.

Misol (ceil va floor uchun):

```
#include<stdio.h>
#include<math.h>
```



```
int main()
{
double number=123.54;
double down, up;
down=floor(number);
up=ceil(number);
printf("number=%5.2lf\n", number);
printf("down=%5.2lf\n", down);
printf("up=%5.2lf\n", up);
return 0;
}
```

Ildiz. Kvadrat ildiz olish uchun sqrt funksiyasidan foydalaniladi.

Prototipi:

double sqrt(double x);

Agar parametr qiymati musbat bo'lsa, natija ham musbat, manfiy bo'lsa, xatolik kelib chiqadi.

Misol:

```
#include<stdio.h>
#include<math.h>
int main()
{
double x=4.0;
double z=sqrt(x);
printf("x=%lf sqrt(x)=%lf\n", x, z);
return 0;
}
```

Modul. Modul, ya'ni ikki son bo'linmasi x/y qoldig'ini hisoblash uchun fmod funksiyasidan foydalaniladi.

Prototipi:

double fmod(double x, double y);

Funksiya qiymati f quyidagi shartlarga mos keladi:

x=(ay+f), bunda a butun son va 0<f<y.

Agar y=0 bo'lsa, fmod qiymati 0 ga teng bo'ladi.

Misol:

```
#include<stdio.h>
```

```
#include<math.h>
int main()
{
double x=5.0, y=2.0;
double z=fmod(x, y);
printf("x=%lf y=%lf fmod(x, y))=%lf\n", x, y, z);
return 0;
}
```

Kasr ajratish. Haqiqiy sonning butun va kasr qismini ajratish uchun modf funksiyasidan foydalaniladi.

Prototipi:

double modf(double x, double *ipart);

Funksiya double x sonning butun qismini ipart parametrga yozib, kasr qismini qiymat sifatida qaytaradi.

Misol:

```
#include<stdio.h>
#include<math.h>
int main()
{
double fraction, integer;
double number=100000.567;
fraction=modf(number, &integer);
printf("number=%lf integer=%lf fraction=%lf\n", number, integer, fraction);
return 0;
}
```

8.2. Simvolli funksiyalar

getchar(arg) makrosi bitta simvol kiritilishini kutish uchun ishlatiladi. Kiritilayotgan simvol monitorda aks etadi.

putchar(arg) makrosi bitta simvolni standart oqimga chiqarish uchun ishlatiladi. Simvol monitorda aks etadi.

Bu funksiyalar **stdio.h** modulida joylashgan.

Prototiplari:

```
int getchar(void);
int putchar(int c);
```


Makros getchar o'rniga standart kiritish oqimidan simvol o'quvchi getc(stdin) funksiyasi joylashtiriladi.

Makros putchar o'rniga standart chiqarish oqimiga simvol yozuvchi putc(c, stdout) funksiyasi joylashtiriladi

Ikkala makros xatolik yuz bersa yoki getchar fayl oxirini o'qisa, EOF qaytaradi.

```
#include <stdio.h>
int main(void)
{
char c;
while ((c=getchar())!='#')
putchar(c);
return 0;
}
```

Natija:

```
aaa#bbb
aaa
```

Satrlarni kiritish va chiqarish

Satrlarni kiritish uchun gets funksiyasidan foydalaniladi.

Funksiya gets satrni stdin standart oqimdan kiritadi.

Prototipi:

char *gets(char *s);

Funksiyalardan foydalanish uchun dasturga <stdio.h> sarlavhali fayl ulash lozim.

Funksiya gets argument s ga ko'rsatkich qaytaradi. Agar fayl oxiriga yetilsa yoki xatolik yuz bersa, null qaytaradi.

Misol:

```
#include <stdio.h>
int main(void)
{
char string[80];
printf("Input a string:");
gets(string);
printf("The string input was: %s\n", string);
return 0;
}
```

Satrlarni chiqarish uchun puts funksiyasidan foydalaniladi.

Funksiya puts satrni stdout standart oqimga chiqaradi (yangi satr simvolini qaytaradi).

Prototipi:

int puts(const char *s);

Funksiya puts manfiy bo'lmagan son qaytaradi. Agar xatolik yuz bersa, EOF qaytaradi.

Misol:

```
#include <stdio.h>
int main(void)
{
char string[]="This is an example output string\n";
puts(string);
return 0;
```

Simvolni tekshirish. Simvolni tekshirish uchun quyidagi makroslardan foydalaniladi:

isalnum, isalpha, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit.

Bu makroslardan foydalanish uchun dasturga <ctype.h> sarlavhali faylni ulash lozim.

Prototiplari:

int isalnum(int c); int islower(int c);
int isalpha(int c); int isprint(int c);
int iscntrl(int c); int isspace(int c);
int isdigit(int c); int isupper(int c);
int isgraph(int c); int isxdigit(int c);

Makroslar argumentlari simvol ASCII kodini bildiruvchi butun son.

Makroslar qiymatlari, agar shart bajarilsa, nolga teng bo'lmagan son va aks holda 0.

Har bir makrosni #undef direktivasi yordamida rad etish mumkin.

isalpha: c harf (A dan Z gacha yoki a dan z gacha)

iscntrl: c delete simvoli yoki control simvol (0x7F yoki 0x00 yoki 0x1F)

isdigit: c raqam (0 dan 9 gacha)

isgraph: c bosiluvchi simvol isprint dan farqli, bo'shliq simvoli ham kiradi.

islower: c kichik harf (a dan z gacha)

isprint: c bosiluvchi simvol (0x20 dan 0x7E gacha)

ispunct: c punktuasiya belgisi (iscntrl yoki isspace)

isspace: c yoki bo'shliq, tab, karetkani o'tkazish return, yangi satr, vertikal tab, yoki formfeed (0x09 to 0x0D, 0x20)

isupper: c katta harf (A to Z)

isxdigit: c o'n oltilik raqam (0 dan 9 gacha, A dan F gacha, a dan f gacha)

Katta harflarni kichik harflarga aylantirish uchun tolower funksiyasidan foydalaniladi.

Prototipi:

int tolower(int ch);

Funksiya ch butun son qiymati (EOF kodi 255) katta harf kodi bo'lsa, mos kichik harf kodini qaytaradi (A dan Z gacha; natija a dan z gacha). Qolgan hollarda simvol o'zgarmaydi.

Misol:

```
#include<stdio.h>
#include <ctype.h>
int main()
{
int i;
char str1[20]="THIS IS A STRING";
for (i=0; str1[i]!='\0'; i++)
{
str1[i]=tolower(str1[i]);
}
printf("%s\n", str1);
return 0;
}
```

Kichik harflarni katta harflarga aylantirish uchun toupper funksiyasidan foydalaniladi.

Prototip:

int toupper(int ch);

Funksiya ch butun son qiymati (EOF kodi 255) kichik harf kodi bo'lsa, mos katta harf kodini qaytaradi (a dan z gacha; natija A dan Z gacha). Qolgan hollarda simvol o'zgarmaydi.

Misol:

```
#include<stdio.h>
#include <ctype.h>
int main()
{
int i;
char str1[20]="this is a string";
for (i=0; str1[i]!='\0'; i++)
{
str1[i]=toupper(str1[i]);
}
printf("%s\n", str1);
return 0;
}
```

8.3. Satrli funksiyalar

Satrli funksiyalardan foydalanish uchun dasturga <string.h> sarlavhali fayl ulash lozim

Satrda simvollar sonini hisoblash uchun strlen funksiyasidan foydalaniladi.

Prototipi:

size_t strlen(const char *s);

Satrdagi simvollar sonini qaytaradi. Satr oxirini bildiruvchi null simvol hisobga kirmaydi.

Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char *string="Borland International";
printf("%d\n", strlen(string));
return 0;
}
```

Satrdan nusxa olish uchun strcpy funksiyasidan foydalaniladi.

Prototipi:

char *strcpy(char *dest, const char *src);

Funkiya strcpy satr simvollarini dest satrga nusxa oladi va dest satr qaytaradi.

Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char string[10];
char *str1="abcdefghi";
strcpy(string, str1);
printf("%s\n", string);
return 0;
}
```

Satrning berilgan sondagi simvollaridan nusxa olish uchun strncpy funksiyasidan foydalaniladi.

Prototipi:

char *strncpy(char *dest, const char *src, size_t maxlen);

Funksiya strncpy satr maxlen dan oshmagan simvollarini dest satrga nusxa oladi va dest satr qaytaradi.

Qaytarilayotgan dest satri satr oxirini bildiruvchi null-simvolga ega bo'lmasligi mumkin, agar src uzunligi maxlen ga teng yoki katta bo'lsa.

Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char string[10];
char *str1=«abcdefghi»;
strncpy(string, str1, 3);
string[3]='\0';
printf("%s\n", string);
return 0;
}
```

Satrlarni ulash uchun strcat funksiyasidan foydalaniladi.

Prototipi:
char *strcat(char *dest, const char *src);
Birinchi dest satr oxiriga src satr simvollarini ulaydi.
Natija uzunligi strlen(dest)+strlen(src).
Satrlar konkatenatsiyasiga ko'rsatkich qaytaradi.
Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char destination[25];
char *blank=" ", *c=«C++», *turbo="Turbo";
strcpy(destination, turbo);
strcat(destination, blank);
strcat(destination, c);
printf("%s\n", destination);
return 0;
}
```

Berilgan sondagi simvollarni ulash uchun strncat funksiyasidan foydalanish lozim.
Prototipi:
char *strncat(char *dest, const char *src, size_t maxlen);
Berilgan dest satr oxiriga src satr maxltn dan oshmagan simvollarini ulaydi va satr oxiriga null simvol qo'shadi.
Natija maksimum uzunligi strlen(dest)+maxlen.
Funksiya dest satr qaytaradi.
Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char destination[25];
char *source="States";
strcpy(destination, "United");
strncat(destination, source, 7);
```



```
printf("%s\n", destination);
return 0;
}
```

Satrlarni solishtirish uchun strcmp funksiyasidan foydalaniladi.
Prototipi:
int strcmp(const char *s1, const char*s2);
funksiya s1 va s2 satrlarni leksikografik solishtiradi.
Qaytaradigan qiymati:
natija<0, agar s1 <s2
natija== 0, agar s1 == s2
natija > 0, agar s1 >s2
Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char *buf1="aaa", *buf2="aab";
int ptr;
ptr=strcmp(buf2, buf1);
if (ptr > 0) printf("buffer 2 > buffer 1\n");
if(ptr<0) printf("buffer 2<buffer 1\n");
if(ptr == 0) printf("buffer 2=buffer 1\n");
return 0;
}
```

Satrlarning berilgan sondagi simvollarini solishtirish uchun strncmp funksiyasidan foydalaniladi.
Prototipi:
int strncmp (const char *s1, const char *s2, size_t maxlen);
Funksiya maxlen dan ko'p bo'lmagan s1va s2 satr simvollarini leksikografik solishtiradi.
Qaytaradigan qiymati:
natija<0, agar s1 <s2
natija== 0, agar s1 == s2
natija > 0, agar s1 > s2
Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char *buf1="aaa", *buf2="aab";
int ptr;
ptr=strncmp(buf2, buf1,2);
if (ptr>0) printf("buffer 2 >buffer 1\n");
if(ptr<0) printf("buffer 2<buffer 1\n");
if(ptr==0) printf("buffer 2=buffer 1\n");
return 0;
}
```

Satrda simvolning birinchi kirishini qidirish uchun strchr funksiyasidan foydalaniladi.

Prototipi:

char *strchr(char *s, int c);

Funksiya s satrda c simvolning birinchi kirishini qidiradi.

Satr oxirini bildiruvchi null-simvolni quyidagicha qidirish mumkin: strchr(strs, 0).

Funksiya simvolning birinchi kirishiga ko'rsatkich qaytaradi. Xatolik yuz bersa, null qaytaradi.

Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char* string="This is a string";
char *ptr, c='i';
ptr=strchr(string, c);
if (ptr)
printf("character=%c position=%d\n", c, ptr-string);
else
printf("character=%c not found\n", c);
return 0;
}
```


Satrda simvolning oxirgi kirishini qidirish uchun strchr funksiyasidan foydalaniladi.

Prototipi:

char *strrchr(char *s1, int c);

Funksiya s satrda c simvolning oxirgi kirishini qidiradi.

Satr oxirini bildiruvchi null-simvolni quyidagicha qidirish mumkin: strchr(strs, 0).

Funksiya simvolning oxirgi kirishiga ko'rsatkich qaytaradi. Xatolik yuz bersa, null qaytaradi.

Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char* string="This is a string";
char *ptr, c='i';
ptr=strrchr(string, c);
if (ptr)
printf("character=%c position=%d\n", c, ptr-string);
else
printf("character=%c not found\n", c);
return 0;
}
```

Satrda ostki satrni izlash uchun strstr funksiyasidan foydalaniladi. Prototipi:

char *strstr(const char *s1, const char *s2);

Funksiya s1 satrda s2 ostki satrning birinchi kirishini izlaydi.

Agar ostki satr mavjud bo'lsa, birinchi kirishiga ko'rsatkich, aks holda null qaytaradi.

Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char *str1="Borland International", *str2="nation", *ptr;
ptr=strstr(str1, str2);
```

```
if (ptr == NULL) printf("NO");
else printf("YES");
return 0;
}
```

Birinchi satrga birinchi kiruvchi ikkinchi satrdagi simvolni topish uchun strpbrk funksiyasidan foydalaniladi.

Prototipi:

char *strpbrk(const char *s1, const char *s2);

s1 satrga birinchi kiruvchi s2 satrdagi simvolni qidiradi. Agar simvol mavjud bo'lsa, shu simvolga ko'rsatkich qaytaradi, aks holda null qaytaradi.

Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char *string1="abcdefghijklmnopqrstuvwxyz";
char *string2="onm";
char *ptr;
ptr=strpbrk(string1, string2);
if (ptr)
printf("first character=%c\n", *ptr);
else
printf("find character not\n");
return 0;
}
```

Ikki satr orasida farqni aniqlash uchun strtok funksiyasidan foydalaniladi.

Prototipi:

char *strtok(char *s1, const char *s2);

Funksiya s1 satrda s2 satrdan farqli bir yoki bir necha simvollarni aniqlaydi.

Birinchi chaqirishda s1 satrdagi farqli birinchi ostki satrga ko'rsatkich qaytaradi. Bunday ostki satr mavjud bo'lmasa, null qaytaradi. Agar qayta

chaqirishda birinchi parametri birinchi NULL bo'lsa, ikkinchi kirishidan oldingi farq qiluvchi ostki satrga ko'rsatkich qaytaradi.

Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char input[16]="adbcetbc";
char *p;
p=strtok(input, "bc");
if (p) printf("%s\n", p);
p=strtok(NULL, "bc");
if (p) printf("%s\n", p);
return 0;
}
```

Bir satr boshida ikkinchi satrdan farqli qismini izlash uchun strcspn funksiyasidan foydalaniladi.

Prototipi:

size_t strcspn(const char *s1, const char *s2);

Birinchi strcspn funksiyasi s1 satr boshidan shunday segment izlaydiki, biror simvoli s2 satrga kirmaydi.

Funksiya boshlang'ich satr uzunligini qaytaradi.

```
#include<stdio.h>
#include <string.h>
int main()
{
char input[16]="cdatbcetbc";
int p;
p=strcspn(input, "abc");
printf("%d\n", p);
return 0;
}
```

Bir satr boshida ikkinchi satr izlash uchun strspn funksiyalaridan foydalaniladi.

Prototipi:

size_t strspn(const char *s1, const char *s2);

Ikkinchi strspn funksiyasi s1 satr boshidan shunday segment izlaydiki, hamma simvollari s2 satrga kiradi.

Funksiya boshlang'ich satr uzunligini qaytaradi.

```
#include<stdio.h>
#include <string.h>
int main()
{
char input[16]="cbdatbcetbc";
int p;
p=strspn(input, "abc");
printf("%d\n", p);
return 0;
}
```

8.4. Satrni songa aylantirish funksiyalari

Quyida ko'riladigan matematik funksiyalardan foydalanish uchun dasturga <stdlib.h > sarlavhali fayl ulash lozim

Butun songa aylantirish. Satr shaklida berilgan butun sonni songa aylantirish uchun atoi makrosidan foydalaniladi.

Prototipi: int atoi(const char *s);

Satr quyidagi shaklda berilgan bo'lishi kerak:

[ws] [sn] [ddd]

Bu yerda

Tabulatsiya yoki bo'shliq belgisi [ws]

Ishora belgisi [sn]

Raqamlar satri [ddd]

Dastlab aylantirib bo'lmaydigan simvolgacha bajariladi.

Misol:

```
#include<stdio.h>
#include <stdlib.h>
int main()
{
int n;
char *str="12345.67";
```



```
n=atoi(str);
printf("string=%s integer=%d\n", str, n);
return 0;
}
```

Uzun songa aylantirish. Satr shaklida berilgan uzun sonni long turidagi son sifatida qaytarish uchun atol funksiyasidan foydalaniladi.

Prototipi: long atol(const char *s);

Satr quyidagi shaklda bo'lishi kerak:

[ws] [sn] [ddd]

Dastlab aylantirib bo'lmaydigan simvolgacha bajariladi. Aylantirish mumkin bo'lmasa, 0 qaytariladi.

Misol:

```
#include<stdio.h>
#include <stdlib.h>
int main()
{
long l;
char *lstr="98765432";
l=atol(lstr);
printf("string=%s integer=%ld\n", lstr, l);
return 0;
}
```

Haqiqiy songa aylantirish. Satr shaklida berilgan haqiqiy sonni son sifatida qaytaruvchi funksiya atof deb nomlanadi.

Prototipi:

double atof(const char *s);

Satr quyidagi shaklda berilgan bo'lishi kerak:

[whitespace] [sign] [ddd] [.] [ddd] [e|E[sign]ddd]

Misol:

```
#include<stdio.h>
#include <stdlib.h>
int main()
{
float f;
char *str="12345.67";
```

```
f=atof(str);
printf("string=%s float=%f\n", str, f);
return 0;
}
```

Satrni ikkilangan haqiqiy songa aylantirish uchun strtod funksiyasidan foydalaniladi.

Prototipi:

double strtod(const char *s, char **endptr);

Bu funksiyadan foydalanish uchun <stdlib.h> sarlavhali faylni ulash lozim.

Satr quyidagi ko'rinishda bo'lishi lozim:

strtod: [ws] [sn] [ddd] [.] [ddd] [fmt[sn]ddd]

bu yerda

[ws]=bo'shliq

[sn]=ishora (+ yoki –)

[ddd]=raqamlar

[fmt]=e yoki E

[.]=nuqta

[0]=nol zero (0)

[x]=x yoki X

Bundan tashqari, +INF va – INF qaytaradi plus yoki minus cheksizlik va +NAN va –NAN qaytaradi Not-A-Number uchun.

Funksiya to aylantirib bo'lmaydigan birinchi simvolgacha ishlaydi.

Misol:

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
char input[80], *endptr;
double value;
printf("Enter a floating point number:");
gets(input);
value=strtod(input, &endptr);
printf("The string is %s the number is %lf\n", input, value);
return 0;
}
```


8-bob bo'yicha savollar

1. Trigonometrik funksiyalarni ko'rsating.
2. Teskari trigonometrik funksiyalarni ko'rsating.
3. Qanday yaxlitlash funksiyalari mavjud?
4. Ixtiyoriy asosdagi logarifm qanday hisoblanadi?
5. Ixtiyoriy ildiz qanday hisoblanadi?
6. Ikki satr orasidagi farqni aniqlash uchun qaysi funksiyalardan foydalaniladi?
7. Satrdan nusxa olish uchun qanday funksiyalardan foydalaniladi?
8. Satrlarni solishtirish uchun qanday funksiyalardan foydalaniladi?
9. Simvolli funksiyalardan foydalanish uchun qanday sarlavhali faylni ulash lozim?
10. Satrni songa aylantirish funksiyalarini ko'rsating.

8-bob bo'yicha topshiriqlar

1. Trigonometrik funksiyalarga dastur yarating.
2. Daraja, eksponenta va logarifm funksiyalariga dastur yarating.
3. Simvolli tekshiruvchi funksiyalarga dastur yarating.
4. Simvolli almashtiruvchi funksiyalarga dastur yarating.
5. Satrni songa aylantiruvchi funksiyalarga dastur yarating.

9-bob. STANDART FUNKSIYALAR

9.1. Dastur bajarilishini boshqarish funksiyalari

Dasturni to'xtatish. Jarayonni to'xtatish uchun abort funksiyasidan foydalaniladi.

Prototipi: void abort(void);

Funksiyadan foydalanish uchun <stdlib.h> sarlavhali faylni ulash lozim.

Funksiya abort stderr oqimga ("Abnormal program termination") ma'lumot yozadi va 3 chiqish kodini hosil qiladi.

Misol:

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
printf("Calling abort()\n");
abort();
return 0; /* This is never reached */
}
```

Dasturdan chiqish. Dasturdan chiqish uchun exit funksiyasidan foydalaniladi.

Prototipi: void exit(int status);

Funksiyadan foydalanish uchun <stdlib.h> sarlavhali faylni ulash lozim.

Normal holda 0 kodi bilan chiqiladi. Aks holda nolga teng bo'lmagan kod bilan chiqiladi. Chiqishdan oldin hamma fayllar berkitiladi.

Funksiya qiymat qaytarmaydi.

Misol:

```
#include <stdio.h>
```



```
#include <stdlib.h>
int main(void)
{
char status;
do {
printf("Enter 1 or 2\n");
status=getchar();}
while((status<'1')||(status>'2'));
exit(status – '0');
/* Note: this line is never reached */
return 0;
}
```

Dastur bajarilishini tekshirish. Dastur bajarilishini tekshirish uchun assert makrosidan foydalaniladi. Makros deb tanasi chaqirig'i o'rniga joylashtiriluvchi funksiyaga aytiladi.

Prototipi:

void assert(int test);

Funksiyadan foydalanish uchun <assert.h> sarlavhali faylni ulash lozim.

Agar tekshirish 0 qaytarsa, assert stderr oqimiga quyidagi ma'lumot chiqariladi.

Assertion failed: test, fayl nomi, qator nomeri.

Shundan so'ng abort funksiyasi chaqirilib, dastur to'xtatiladi.

Agar #include <assert.h> direktivasi oldidan #define NDEBUG directive ("no debugging") direktivasi qo'yilsa, keyingi funksiyalar bajarilmaydi.

Funksiya qiymat qaytarmaydi.

Misol:

```
#include <stdio.h>
#include <assert.h>
#include <string.h>
struct ITEM {
int key;
int value;
};
```

```
/* add item to list, make sure list is not null */
void additem(struct ITEM *itemptr) {
assert(itemptr !=NULL);
/* add item to list */
}

int main(void)
{
additem(NULL);
return 0;
}
```

Kutilmagan hodisalar. Dastur bajarilishi jarayoni kutilmagan hodisalar, uzilishlar haqida ma'lumot berish uchun raise funksiyasidan foydalaniladi.

Prototipi: int raise(int sig);

Funksiyadan foydalanish uchun <signal.h> sarlavhali faylni ulash lozim.

Dastur bajarilishi jarayonida hosil bo'lgan kutilmagan hodisalar uzilishlar haqida ma'lumot berish uchun foydalaniladi.

Agar xatolik bo'lmasa, 0 qaytaradi. Aks holda, 0 ga teng bo'lmagan son qaytaradi.

Misol:

```
#include <signal.h>
int main(void)
{
int a, b;
a=10;
b=0;
if (b == 0)
/* preempt divide by zero error */
raise(SIGFPE);
else a=a / b;
return 0;
}
```

SIGFPE arifmetik xatolikni bildirib, exit(1) funksiyani chaqiradi.

9.2. Tasodifiy sonlar va vaqt

Tasodifiy sonlar. Tasodifiy sonlar generatori rand deb ataladi.

Sarlavhasi: int rand(void);

Funksiyadan foydalanish uchun <stdlib.h> sarlavhali faylni ulash lozim.

Funksiya rand davri 232 bo'lgan tasodifiy sonlar generatoridan foydalangan holda 0 dan to RAND_MAX oraliqdagi tasodifiy son qaytaradi.

Misol. 0 va 99 oraliqdagi tasodifiy sonlar generatsiyasi:

```
#include<stdio.h>
#include <stdlib.h>
#include<conio.h>
int main()
{
int i;
for(i=0; i<10; i++)
printf("%d\n", rand() % 100);
getch();
return 0;
}
```

Tasodifiy sonlar generatorini initsializatsiya qilish uchun srand funksiyasidan foydalaniladi.

Sarlavhasi: void srand(unsigned seed);

Funksiyadan foydalanish uchun <stdlib.h> sarlavhali faylni ulash lozim.

Tasodifiy sonlar generatori qayta initsializatsiya qilinadi, agar srand funksiyasi 1 qiymat bilan chaqirilsa.

Funksiya qiymat qaytarmaydi.

Misol:

```
#include<stdio.h>
#include <stdlib.h>
#include<conio.h>
int main()
{
int i;
```

```
for(i=0; i<10; i++)
printf("%d\n", rand() % 100);
printf("\n");
srand(1);
for(i=0; i<10; i++)
printf("%d\n", rand() % 100);
getch();
return 0;
}
```

Vaqt. Vaqtni olish uchun time funksiyasidan foydalaniladi.

Sarlavhasi: time_t time(time_t *timer);

Funksiyadan foydalanish uchun dasturga <time.h> sarlavhali faylni ulash lozim.

Funksiya time vaqtni sekundlarda 00:00:00 GMT, January 1, 1970 dan boshlab oladi.

Funksiya vaqtni sekundlarda qaytaradi. Xatolik yuz bersa, 0 qaytaradi.

Misol:

```
#include<stdio.h>
#include <time.h>
int main()
{int i;
time_t t1, t2;
t1=time(NULL);
printf("%ld\n", t1);
for(i=0; i<1000000000; i++);
t2=time(NULL);
printf("%ld\n", t2);
printf("%ld", t2-t1);
return 0;
}
```

9.3. Xotira ajratish funksiyalari

Dinamik xotira ajratish. Dinamik xotira ajratish uchun C tilida **malloc** funksiyasidan foydalanish lozim. Bu funksiya argumenti ajratilishi lozim

bo'lgan baytlar soni bo'lib, *void turidagi ajratilgan xotiraga ko'rsatkich qaytaradi.

Funksiya prototipi:

void *malloc(size_t size);

Funksiya uyumli xotiradan (size) baytga teng blok ajratadi. Agar ajratish muvaffaqiyatli bo'lsa, ajratilgan xotiraga ko'rsatkich qaytaradi, aks holda (blok ajratish mumkin bo'lmasa yoki hajmi 0 bo'lsa), noll qaytaradi.

Agar ajratilgan xotira bo'shatilmasa, bu xotira foydalanilmay qoladi, agar uning ko'rsatkichi o'z qiymatini o'zgartirgan bo'lsa, uni tizimga qaytarish mumkin emas. Bu hodisa xotiraning yo'qotilishi degan maxsus nom bilan ataladi. Pirovard natijada, dastur xotira yetishmagani tufayli avariya holatida tugallanadi (agar u ancha vaqt ishlayversa).

Dinamik ajratilgan xotirani bo'shatish uchun free funksiyasidan foydalanish lozim

Funksiya prototipi:

void free(void *block);

Shuni hisobga olish lozimki, bu funksiya yordamida bo'shatilgan xotirani yana bo'shatish mumkin emas.

Misol:

```
#include <stdio.h>
#include<stdlib.h>
#include <string.h>
int main(void)
{
char *str;
/* allocate memory for string */
if ((str=(char *) malloc(10))== NULL)
{
printf("Not enough memory to allocate buffer\n");
exit(1); /* terminate program if out of memory */
}
/* copy "Hello" into string */
strcpy(str, "Hello");
/* display string */
printf("String is %s\n", str);
/* free memory */
```

```
free(str);
return 0;
}
```

Xotira ajratish funksiyalari. Xotira ajratib, initsiallash uchun calloc funksiyasidan foydalaniladi.

Bu funksiyadan foydalanish uchun <stdlib.h> sarlavhali faylni ulash lozim.

Prototipi: void *calloc(size_t nitems, size_t size);

Funksiya calloc uyumli xotiradan (nitems * size) baytga teng blok ajratadi va 0 qiymat bilan to'ldiradi. Blok 64K dan oshmasligi lozim. Agar ajratish muvaffaqiyatli bo'lsa, ajratilgan xotiraga ko'rsatkich qaytaradi, aks holda (blok ajratish mumkin bo'lmasa yoki hajmi 0 bo'lsa), noll qaytaradi.

Misol:

```
#include <stdio.h>
#include<stdlib.h>
#include <string.h>
int main(void)
{
char *str=NULL;
/* allocate memory for string */
str=(char *) calloc(10, sizeof(char));
/* copy "Hello" into string */
strcpy(str, "Hello");
/* display string */
printf("String is %s\n", str);
/* free memory */
free(str);
return 0;
}
```

Xotirani qayta ajratish. Xotirani qayta ajratish uchun realloc funksiyasidan foydalaniladi. Bu funksiyadan foydalanish uchun <stdlib.h> sarlavhali faylni ulash lozim.

Prototipi:

void *realloc(void *block, size_t size);

Funksiya realloc ajratilgan xotira hajmini o'zgartiradi. Kerak bo'lsa, yangi joyga nusxa oladi.


```
printf("%s\n", destination);
return 0;
}
```

Satrlarni solishtirish uchun strcmp funksiyasidan foydalaniladi.

Prototipi:

int strcmp(const char *s1, const char*s2);

funksiya s1 va s2 satrlarni leksikografik solishtiradi.

Qaytaradigan qiymati:

natija<0, agar s1 <s2

natija== 0, agar s1 == s2

natija > 0, agar s1 >s2

Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char *buf1="aaa", *buf2="aab";
int ptr;
ptr=strcmp(buf2, buf1);
if (ptr > 0) printf("buffer 2 > buffer 1\n");
if(ptr<0) printf("buffer 2<buffer 1\n");
if(ptr == 0) printf("buffer 2=buffer 1\n");
return 0;
}
```

Satrlarning berilgan sondagi simvollarini solishtirish uchun strncmp funksiyasidan foydalaniladi.

Prototipi:

int strncmp (const char *s1, const char *s2, size_t maxlen);

Funksiya maxlen dan ko'p bo'lmagan s1va s2 satr simvollarini leksikografik solishtiradi.

Qaytaradigan qiymati:

natija<0, agar s1 <s2

natija== 0, agar s1 == s2

natija > 0, agar s1 > s2

Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char *buf1="aaa", *buf2="aab";
int ptr;
ptr=strncmp(buf2, buf1,2);
if (ptr>0) printf("buffer 2 >buffer 1\n");
if(ptr<0) printf("buffer 2<buffer 1\n");
if(ptr==0) printf("buffer 2=buffer 1\n");
return 0;
}
```

Satrda simvolning birinchi kirishini qidirish uchun strchr funksiyasidan foydalaniladi.

Prototipi:

char *strchr(char *s, int c);

Funksiya s satrda c simvolning birinchi kirishini qidiradi.

Satr oxirini bildiruvchi null-simvolni quyidagicha qidirish mumkin: strchr(strs, 0).

Funksiya simvolning birinchi kirishiga ko'rsatkich qaytaradi. Xatolik yuz bersa, null qaytaradi.

Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char* string="This is a string";
char *ptr, c='i';
ptr=strchr(string, c);
if (ptr)
printf("character=%c position=%d\n", c, ptr-string);
else
printf("character=%c not found\n", c);
return 0;
}
```


Satrda simvolning oxirgi kirishini qidirish uchun strchr funksiyasidan foydalaniladi.

Prototipi:

char *strrchr(char *s1, int c);

Funksiya s satrda c simvolning oxirgi kirishini qidiradi.

Satr oxirini bildiruvchi null-simvolni quyidagicha qidirish mumkin: strchr(strs, 0).

Funksiya simvolning oxirgi kirishiga ko'rsatkich qaytaradi. Xatolik yuz bersa, null qaytaradi.

Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char* string="This is a string";
char *ptr, c='i';
ptr=strrchr(string, c);
if (ptr)
printf("character=%c position=%d\n", c, ptr-string);
else
printf("character=%c not found\n", c);
return 0;
}
```

Satrda ostki satrni izlash uchun strstr funksiyasidan foydalaniladi. Prototipi:

char *strstr(const char *s1, const char *s2);

Funksiya s1 satrda s2 ostki satrning birinchi kirishini izlaydi.

Agar ostki satr mavjud bo'lsa, birinchi kirishiga ko'rsatkich, aks holda null qaytaradi.

Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char *str1="Borland International", *str2="nation", *ptr;
ptr=strstr(str1, str2);
```

```
if (ptr == NULL) printf("NO");
else printf("YES");
return 0;
}
```

Birinchi satrga birinchi kiruvchi ikkinchi satrdagi simvolni topish uchun strpbrk funksiyasidan foydalaniladi.

Prototipi:

char *strpbrk(const char *s1, const char *s2);

s1 satrga birinchi kiruvchi s2 satrdagi simvolni qidiradi. Agar simvol mavjud bo'lsa, shu simvolga ko'rsatkich qaytaradi, aks holda null qaytaradi.

Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char *string1="abcdefghijklmnopqrstuvwxyz";
char *string2="onm";
char *ptr;
ptr=strpbrk(string1, string2);
if (ptr)
printf("first character=%c\n", *ptr);
else
printf("find character not\n");
return 0;
}
```

Ikki satr orasida farqni aniqlash uchun strtok funksiyasidan foydalaniladi.

Prototipi:

char *strtok(char *s1, const char *s2);

Funksiya s1 satrda s2 satrdan farqli bir yoki bir necha simvollarni aniqlaydi.

Birinchi chaqirishda s1 satrdagi farqli birinchi ostki satrga ko'rsatkich qaytaradi. Bunday ostki satr mavjud bo'lmasa, null qaytaradi. Agar qayta

chaqirishda birinchi parametri birinchi NULL bo'lsa, ikkinchi kirishidan oldingi farq qiluvchi ostki satrga ko'rsatkich qaytaradi.

Misol:

```
#include<stdio.h>
#include <string.h>
int main()
{
char input[16]="adbcctbc";
char *p;
p=strtok(input, "bc");
if (p) printf("%s\n", p);
p=strtok(NULL, "bc");
if (p) printf("%s\n", p);
return 0;
}
```

Bir satr boshida ikkinchi satrdan farqli qismini izlash uchun strcspn funksiyasidan foydalaniladi.

Prototipi:

size_t strcspn(const char *s1, const char *s2);

Birinchi strcspn funksiyasi s1 satr boshidan shunday segment izlaydiki, biror simvoli s2 satrga kirmaydi.

Funksiya boshlang'ich satr uzunligini qaytaradi.

```
#include<stdio.h>
#include <string.h>
int main()
{
char input[16]="cdatbcctbc";
int p;
p=strcspn(input, "abc");
printf("%d\n", p);
return 0;
}
```

Bir satr boshida ikkinchi satr izlash uchun strspn funksiyalaridan foydalaniladi.

Prototipi:

size_t strspn(const char *s1, const char *s2);

Ikkinchi strspn funksiyasi s1 satr boshidan shunday segment izlaydiki, hamma simvollari s2 satrga kiradi.

Funksiya boshlang'ich satr uzunligini qaytaradi.

```
#include<stdio.h>
#include <string.h>
int main()
{
char input[16]="cbdatbcetbc";
int p;
p=strspn(input, "abc");
printf("%d\n", p);
return 0;
}
```

8.4. Satrni songa aylantirish funksiyalari

Quyida ko'riladigan matematik funksiyalardan foydalanish uchun dasturga <stdlib.h > sarlavhali fayl ulash lozim

Butun songa aylantirish. Satr shaklida berilgan butun sonni songa aylantirish uchun atoi makrosidan foydalaniladi.

Prototipi: int atoi(const char *s);

Satr quyidagi shaklda berilgan bo'lishi kerak:

[ws] [sn] [ddd]

Bu yerda

Tabulatsiya yoki bo'shliq belgisi [ws]

Ishora belgisi [sn]

Raqamlar satri [ddd]

Dastlab aylantirib bo'lmaydigan simvolgacha bajariladi.

Misol:

```
#include<stdio.h>
#include <stdlib.h>
int main()
{
int n;
char *str="12345.67";
```



```
n=atoi(str);
printf("string=%s integer=%d\n", str, n);
return 0;
}
```

Uzun songa aylantirish. Satr shaklida berilgan uzun sonni long turidagi son sifatida qaytarish uchun atol funksiyasidan foydalaniladi.

Prototipi: long atol(const char *s);

Satr quyidagi shaklda bo'lishi kerak:

[ws] [sn] [ddd]

Dastlab aylantirib bo'lmaydigan simvolgacha bajariladi. Aylantirish mumkin bo'lmasa, 0 qaytariladi.

Misol:

```
#include<stdio.h>
#include <stdlib.h>
int main()
{
long l;
char *lstr="98765432";
l=atol(lstr);
printf("string=%s integer=%ld\n", lstr, l);
return 0;
}
```

Haqiqiy songa aylantirish. Satr shaklida berilgan haqiqiy sonni son sifatida qaytaruvchi funksiya atof deb nomlanadi.

Prototipi:

double atof(const char *s);

Satr quyidagi shaklda berilgan bo'lishi kerak:

[whitespace] [sign] [ddd] [.] [ddd] [e|E[sign]ddd]

Misol:

```
#include<stdio.h>
#include <stdlib.h>
int main()
{
float f;
char *str="12345.67";
```

```
f=atof(str);
printf("string=%s float=%f\n", str, f);
return 0;
}
```

Satrni ikkilangan haqiqiy songa aylantirish uchun strtod funksiyasidan foydalaniladi.

Prototipi:

double strtod(const char *s, char **endptr);

Bu funksiyadan foydalanish uchun <stdlib.h> sarlavhali faylni ulash lozim.

Satr quyidagi ko'rinishda bo'lishi lozim:

strtod: [ws] [sn] [ddd] [.] [ddd] [fmt[sn]ddd]

bu yerda

[ws]=bo'shliq

[sn]=ishora (+ yoki –)

[ddd]=raqamlar

[fmt]=e yoki E

[.]=nuqta

[0]=nol zero (0)

[x]=x yoki X

Bundan tashqari, +INF va – INF qaytaradi plus yoki minus cheksizlik va +NAN va –NAN qaytaradi Not-A-Number uchun.

Funksiya to aylantirib bo'lmaydigan birinchi simvolgacha ishlaydi.

Misol:

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
char input[80], *endptr;
double value;
printf("Enter a floating point number:");
gets(input);
value=strtod(input, &endptr);
printf("The string is %s the number is %lf\n", input, value);
return 0;
}
```


8-bob bo'yicha savollar

1. Trigonometrik funksiyalarni ko'rsating.
2. Teskari trigonometrik funksiyalarni ko'rsating.
3. Qanday yaxlitlash funksiyalari mavjud?
4. Ixtiyoriy asosdagi logarifm qanday hisoblanadi?
5. Ixtiyoriy ildiz qanday hisoblanadi?
6. Ikki satr orasidagi farqni aniqlash uchun qaysi funksiyalardan foydalaniladi?
7. Satrdan nusxa olish uchun qanday funksiyalardan foydalaniladi?
8. Satrlarni solishtirish uchun qanday funksiyalardan foydalaniladi?
9. Simvolli funksiyalardan foydalanish uchun qanday sarlavhali faylni ulash lozim?
10. Satrni songa aylantirish funksiyalarini ko'rsating.

8-bob bo'yicha topshiriqlar

1. Trigonometrik funksiyalarga dastur yarating.
2. Daraja, eksponenta va logarifm funksiyalariga dastur yarating.
3. Simvolli tekshiruvchi funksiyalarga dastur yarating.
4. Simvolli almashtiruvchi funksiyalarga dastur yarating.
5. Satrni songa aylantiruvchi funksiyalarga dastur yarating.

9-bob. STANDART FUNKSIYALAR

9.1. Dastur bajarilishini boshqarish funksiyalari

Dasturni to'xtatish. Jarayonni to'xtatish uchun abort funksiyasidan foydalaniladi.

Prototipi: void abort(void);

Funksiyadan foydalanish uchun <stdlib.h> sarlavhali faylni ulash lozim.

Funksiya abort stderr oqimga ("Abnormal program termination") ma'lumot yozadi va 3 chiqish kodini hosil qiladi.

Misol:

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
printf("Calling abort()\n");
abort();
return 0; /* This is never reached */
}
```

Dasturdan chiqish. Dasturdan chiqish uchun exit funksiyasidan foydalaniladi.

Prototipi: void exit(int status);

Funksiyadan foydalanish uchun <stdlib.h> sarlavhali faylni ulash lozim.

Normal holda 0 kodi bilan chiqiladi. Aks holda nolga teng bo'lmagan kod bilan chiqiladi. Chiqishdan oldin hamma fayllar berkitiladi.

Funksiya qiymat qaytarmaydi.

Misol:

```
#include <stdio.h>
```



```
#include <stdlib.h>
int main(void)
{
char status;
do {
printf("Enter 1 or 2\n");
status=getchar();}
while((status<'1')||(status>'2'));
exit(status – '0');
/* Note: this line is never reached */
return 0;
}
```

Dastur bajarilishini tekshirish. Dastur bajarilishini tekshirish uchun assert makrosidan foydalaniladi. Makros deb tanasi chaqirig'i o'rniga joylashtiriluvchi funksiyaga aytiladi.

Prototipi:

void assert(int test);

Funksiyadan foydalanish uchun<assert.h> sarlavhali faylni ulash lozim.

Agar tekshirish 0 qaytarsa, assert stderr oqimiga quyidagi ma'lumot chiqariladi.

Assertion failed: test, fayl nomi, qator nomeri.

Shundan so'ng abort funksiyasi chaqirilib, dastur to'xtatiladi.

Agar #include <assert.h> direktivasi oldidan #define NDEBUG directive ("no debugging") direktivasi qo'yilsa, keyingi funksiyalar bajarilmaydi.

Funksiya qiymat qaytarmaydi.

Misol:

```
#include <stdio.h>
#include <assert.h>
#include <string.h>
struct ITEM {
int key;
int value;
};
```

```
/* add item to list, make sure list is not null */
void additem(struct ITEM *itemptr) {
assert(itemptr != NULL);
/* add item to list */
}

int main(void)
{
additem(NULL);
return 0;
}
```

Kutilmagan hodisalar. Dastur bajarilishi jarayoni kutilmagan hodisalar, uzilishlar haqida ma'lumot berish uchun raise funksiyasidan foydalaniladi.

Prototipi: int raise(int sig);

Funksiyadan foydalanish uchun <signal.h> sarlavhali faylni ulash lozim.

Dastur bajarilishi jarayonida hosil bo'lgan kutilmagan hodisalar uzilishlar haqida ma'lumot berish uchun foydalaniladi.

Agar xatolik bo'lmasa, 0 qaytaradi. Aks holda, 0 ga teng bo'lmagan son qaytaradi.

Misol:

```
#include <signal.h>
int main(void)
{
int a, b;
a = 10;
b = 0;
if (b == 0)
/* preempt divide by zero error */
raise(SIGFPE);
else a = a / b;
return 0;
}
```

SIGFPE arifmetik xatolikni bildirib, exit(1) funksiyani chaqiradi.

9.2. Tasodifiy sonlar va vaqt

Tasodifiy sonlar. Tasodifiy sonlar generatori rand deb ataladi.

Sarlavhasi: int rand(void);

Funksiyadan foydalanish uchun <stdlib.h> sarlavhali faylni ulash lozim.

Funksiya rand davri 232 bo'lgan tasodifiy sonlar generatoridan foydalangan holda 0 dan to RAND_MAX oraliqdagi tasodifiy son qaytaradi.

Misol. 0 va 99 oraliqdagi tasodifiy sonlar generatsiyasi:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int main()
{
int i;
for(i=0; i<10; i++)
printf("%d\n", rand() % 100);
getch();
return 0;
}
```

Tasodifiy sonlar generatorini initsializatsiya qilish uchun srand funksiyasidan foydalaniladi.

Sarlavhasi: void srand(unsigned seed);

Funksiyadan foydalanish uchun <stdlib.h> sarlavhali faylni ulash lozim.

Tasodifiy sonlar generatori qayta initsializatsiya qilinadi, agar srand funksiyasi 1 qiymat bilan chaqirilsa.

Funksiya qiymat qaytarmaydi.

Misol:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int main()
{
int i;
```

```
for(i=0; i<10; i++)
printf("%d\n", rand() % 100);
printf("\n");
srand(1);
for(i=0; i<10; i++)
printf("%d\n", rand() % 100);
getch();
return 0;
}
```

Vaqt. Vaqtni olish uchun time funksiyasidan foydalaniladi.

Sarlavhasi: time_t time(time_t *timer);

Funksiyadan foydalanish uchun dasturga <time.h> sarlavhali faylni ulash lozim.

Funksiya time vaqtni sekundlarda 00:00:00 GMT, January 1, 1970 dan boshlab oladi.

Funksiya vaqtni sekundlarda qaytaradi. Xatolik yuz bersa, 0 qaytaradi.

Misol:

```
#include<stdio.h>
#include <time.h>
int main()
{int i;
time_t t1, t2;
t1=time(NULL);
printf("%ld\n", t1);
for(i=0; i<1000000000; i++);
t2=time(NULL);
printf("%ld\n", t2);
printf("%ld", t2-t1);
return 0;
}
```

9.3. Xotira ajratish funksiyalari

Dinamik xotira ajratish. Dinamik xotira ajratish uchun C tilida **malloc** funksiyasidan foydalanish lozim. Bu funksiya argumenti ajratilishi lozim

bo'lgan baytlar soni bo'lib, *void turidagi ajratilgan xotiraga ko'rsatkich qaytaradi.

Funksiya prototipi:

void *malloc(size_t size);

Funksiya uyumli xotiradan (size) baytga teng blok ajratadi. Agar ajratish muvaffaqiyatli bo'lsa, ajratilgan xotiraga ko'rsatkich qaytaradi, aks holda (blok ajratish mumkin bo'lmasa yoki hajmi 0 bo'lsa), noll qaytaradi.

Agar ajratilgan xotira bo'shatilmasa, bu xotira foydalanilmay qoladi, agar uning ko'rsatkichi o'z qiymatini o'zgartirgan bo'lsa, uni tizimga qaytarish mumkin emas. Bu hodisa xotiraning yo'qotilishi degan maxsus nom bilan ataladi. Pirovard natijada, dastur xotira yetishmagani tufayli avariya holatida tugallanadi (agar u ancha vaqt ishlayversa).

Dinamik ajratilgan xotirani bo'shatish uchun free funksiyasidan foydalanish lozim

Funksiya prototipi:

void free(void *block);

Shuni hisobga olish lozimki, bu funksiya yordamida bo'shatilgan xotirani yana bo'shatish mumkin emas.

Misol:

```
#include <stdio.h>
#include<stdlib.h>
#include <string.h>
int main(void)
{
char *str;
/* allocate memory for string */
if ((str=(char *) malloc(10))== NULL)
{
printf("Not enough memory to allocate buffer\n");
exit(1); /* terminate program if out of memory */
}
/* copy "Hello" into string */
strcpy(str, "Hello");
/* display string */
printf("String is %s\n", str);
/* free memory */
```

```
free(str);
return 0;
}
```

Xotira ajratish funksiyalari. Xotira ajratib, initsiallash uchun calloc funksiyasidan foydalaniladi.

Bu funksiyadan foydalanish uchun <stdlib.h> sarlavhali faylni ulash lozim.

Prototipi: void *calloc(size_t nitems, size_t size);

Funksiya calloc uyumli xotiradan (nitems * size) baytga teng blok ajratadi va 0 qiymat bilan to'ldiradi. Blok 64K dan oshmasligi lozim. Agar ajratish muvaffaqiyatli bo'lsa, ajratilgan xotiraga ko'rsatkich qaytaradi, aks holda (blok ajratish mumkin bo'lmasa yoki hajmi 0 bo'lsa), noll qaytaradi.

Misol:

```
#include <stdio.h>
#include<stdlib.h>
#include <string.h>
int main(void)
{
char *str=NULL;
/* allocate memory for string */
str=(char *) calloc(10, sizeof(char));
/* copy "Hello" into string */
strcpy(str, "Hello");
/* display string */
printf("String is %s\n", str);
/* free memory */
free(str);
return 0;
}
```

Xotirani qayta ajratish. Xotirani qayta ajratish uchun realloc funksiyasidan foydalaniladi. Bu funksiyadan foydalanish uchun <stdlib.h> sarlavhali faylni ulash lozim.

Prototipi:

void *realloc(void *block, size_t size);

Funksiya realloc ajratilgan xotira hajmini o'zgartiradi. Kerak bo'lsa, yangi joyga nusxa oladi.


```
void delete(struct slist_node*p)
{
struct slist_node* p1=p->next;
if (p1!=NULL)
{
p->next=p1->next;
free(p1);
}
}

void insert(struct slist_node*p, int nn)
{
struct slist_node* p1=(struct slist_node*)malloc(sizeof(struct slist_
node));
p1->next=p->next;
p1->info=nn;
p->next=p1;
}

struct slist_node* creat_slist(int size)
{
int i, val;
struct slist_node*beg;
struct slist_node* p;
beg=(struct slist_node*)malloc(sizeof(struct slist_node));
beg->next=NULL;
p=beg;
for(i=0; i<size; i++)
{
scanf("\n%d", &val);
insert(p, val);
p=p->next;
}
return beg;
};

void free_slist(struct slist_node* beg)
```

```
{
struct slist_node* p=beg;
struct slist_node* p1;
while(p!=NULL)
{
p1=p;
p=p1->next;
free(p1);
}
};

void print_slist(struct slist_node* beg)
{
struct slist_node* p=beg->next;
while(p!=NULL)
{
printf("\n%d", p->info);
p=p->next;
}
};
void filtr1_slist(struct slist_node* beg)
{
int val;
struct slist_node* p=beg;
struct slist_node* p1;
while(p->next!=NULL)
{
p1=p->next;
val=p1->info;
if(val%2 == 0) delete(p);
else p=p1;
}
};
int main()
{
struct slist_node* beg;
beg=creat_slist(4);
```



```
print_slist(beg);
printf("\n");
filtr1_slist(beg);
print_slist(beg);
free_slist(beg);
getch();
return 0;
}
```

Bir bog'lamli universal ro'yxat. Universal ro'yxatning axborot qismi void turidagi ko'rsatkichdan iborat bo'ladi:

```
struct slist_node
{
void* info;
struct slist_node* next;
};
```

Asosiy funksiyalardan yangi element qo'shish funksiyasi quyidagi sarlavhaga ega:

void insert(struct slist_node*p, void* val, int width)

Bu yerda val kiritilayotgan ma'lumotga ko'rsatkich va width ma'lumot hajmi.

Dastur matni:

```
#include<stdio.h>
#include<stdlib.h>
#include<mem.h>
#include<conio.h>
struct slist_node
{
void* info;
struct slist_node* next;
};

void delete(struct slist_node*p)
{
struct slist_node* p1=p->next;
if (p1!=NULL)
```

```
    {
    p->next=pl->next;
    free(pl->info);
    free(pl);
    }
    }

    void insert(struct slist_node*p, void* val, int width)
    {
    struct slist_node* pl=(struct slist_node*) malloc(sizeof(struct slist_
node));
    pl->next=p->next;
    pl->info=malloc(width);
    memcpy(pl->info, val, width);
    p->next=pl;
    }

    struct slist_node* creat_slist(int size)
    {
    int i, val;
    struct slist_node*beg;
    struct slist_node* p;
    beg=(struct slist_node*)malloc(sizeof(struct slist_node));
    beg->next=NULL;
    p=beg;
    for(i=0; i<size; i++)
    {
    scanf("\n%d", &val);
    insert(p, (void*)&val, sizeof(int));
    p=p->next;
    }
    return beg;
    };
    void free_slist(struct slist_node* beg)
    {
    struct slist_node* p=beg;
    struct slist_node* pl;
```



```
    while(p!=NULL)
    {
    pl=p;
    p=pl->next;
    free(pl);
    }
    };

    void print_slist(struct slist_node* beg)
    {
    struct slist_node* p=beg->next;
    int*pn;
    while(p!=NULL)
    {
    pn=(int*)p->info;
    printf("\n%d", *pn);
    p=p->next;
    }
    };

    int main()
    {
    struct slist_node* beg;
    beg=creat_slist(4);
    print_slist(beg);
    printf("\n");
    print_slist(beg);
    free_slist(beg);
    getch();
    return 0;
    }
```

10.2. Universal stek va navbat

Stek. Stek deb shunday strukturaga aytiladiki, stekka kelib tushgan oxirgi elementga birinchi bo'lib xizmat ko'rsatiladi va stekdan chiqariladi. Mazkur ko'rinishdagi xizmat ko'rsatishni **LIFO** (*Last input-First output*,

ya'ni oxirgi kelgan – birinchi ketadi) deb nomlash qabul qilingan. Stek bir tomondan ochiq bo'ladi.

Universal stek har bir tugunni axborot qismi void turidagi ko'rsatkichdan iborat strukturadir:

```
struct slist_node
{
void* info;
struct slist_node* pred;
};
```

Stek tugunning ro'yxat tugunidan farqi shundaki, unda o'zidan oldingi tugun adresini saqlovchi ko'rsatkich ishlatilgan.

Stek o'zi alohida struktura sifatida kiritilgan:

```
struct stack
{
struct slist_node* end;
int size;
int width;
};
```

Bu yerda eng oxirgi tugunga ko'rsatkich, width ma'lumot hajmi, size navbatdagi elementlar soni.

Asosiy funksiyalar:

void pop(struct stack*p) – stek oxiridagi elementni o'chirish;

void push(struct stack*p, void* val) – stek oxiriga element qo'shish. Bu yerda val kiritilayotgan ma'lumotga ko'rsatkich;

char* top(struct stack p) – stek oxiridagi tugun axborot qismiga ko'rsatkich qaytarish;

int empty(struct stack p) – stek bo'shligini tekshirish;

int size (struct stack p) – stek elementlari soni.

Bundan tashqari stekni initsiallash uchun quyidagi sarlavhali funksiya kiritilgan.

void ini_stack (struct stack* p, int n) – bu yerda n kiritilayotgan ma'lumotlar hajmi.

Dastur matni:

```
#include<stdio.h>
#include<stdlib.h>
```



```
#include<mem.h>
#include<conio.h>
struct slist_node
{
void* info;
struct slist_node* pred;
};

struct stack
{
struct slist_node* end;
int size;
int width;
};

void pop(struct stack*p)
{
struct slist_node* p1=p->end;
if (p->size>0)
{
p->end=p1->pred;
free(p1);
p->size--;
}
}

void push(struct stack*p, void* val)
{
struct slist_node* p1=(struct slist_node*)malloc(sizeof(struct slist_
node));
p1->info=malloc(p->width);
memcpy(p1->info, val, p->width);
if(p->size==0)
{
p->end=p1;
}
else
```

```
{
pl->pred=p->end;
p->end=pl;
}
p->size++;
}

char* top(struct stack p)
{

return p.end->info;
}

int size(const struct stack* p)
{
return p->size;
}

int empty(struct stack p)
{
if (p.size>0) return 0; else return 1;
}

void ini_stack(struct stack* p, int n)
{
p->end=NULL;
p->size=0;
p->width=n;
}

int main()
{
int i, m;
int*pi;
struct stack ps;
ini_stack(&ps, sizeof(int));
for(i=0; i<5; i++) push(&ps, &i);
```



```
while(!empty(ps))
{
pi=(int*)top(ps);
printf("%d", *pi);
pop(&ps);
}
getch();
return 0;
}
```

Navbat. Navbat deb shunday strukturaga aytiladiki, navbatga kelib tushgan birinchi elementga birinchi bo'lib xizmat ko'rsatiladi va navbatdan chiqariladi. Mazkur ko'rinishdagi xizmat ko'rsatishni **FIFO** (*First input-First output*, ya'ni birinchi kelgan – birinchi ketadi) deb nomlash qabul qilingan. Navbat har ikkala tomondan ochiq bo'ladi.

Universal navbat har bir tuguni axborot qismi void turidagi ko'rsatkichdan iborat strukturadir:

```
struct slist_node
{
void* info;
struct slist_node* next;
};
```

Navbat o'zi alohida struktura sifatida kiritilgan.

```
struct que
{

struct slist_node* beg;
struct slist_node* end;
int size;
int width;
};
```

Bu yerda beg birinchi tugunga ko'rsatkich, end oxirgi tugunga ko'rsatkich, width ma'lumot hajmi, size navbatdagi elementlar soni.

Asosiy funksiyalar:

void pop(struct que*p) – navbat oxiridagi elementni o'chirish;

void push(struct que*p, void* val) – navbat boshiga element qo'shish. Bu yerda val kiritilayotgan ma'lumotga ko'rsatkich;

char* top(struct que p) – navbat boshidagi tugunning axborot qismiga ko'rsatkich qaytarish;

int empty(struct que p) – navbat bo'shligini tekshirish;

int size (struct que p) – navbat elementlari soni.

Bundan tashqari navbatni initsiallash uchun quyidagi sarlavhali funksiya kiritilgan:

void ini_que(struct que* p, int n) – bu yerda n kiritilayotgan ma'lumotlar hajmi.

Dastur matni:

```
#include<stdio.h>
#include<stdlib.h>
#include<mem.h>
#include<conio.h>
struct slist_node
{
void* info;
struct slist_node* next;
};
struct que
{struct slist_node* beg;
struct slist_node* end;
int size;
int width;
};
void pop(struct que*p)
{
struct slist_node* p1=p->beg;
if (p->size>0)
{
p->beg=p1->next;
free(p1);
p->size--;
}
}
void push(struct que*p, void* val)
{
```



```
struct slist_node* p1=(struct slist_node*)malloc(sizeof(struct slist_
node));
p1->info=malloc(p->width);
memcpy(p1->info, val, p->width);
if(p->size==0)
{
p->beg=p1;
p->end=p1;
}
else
{
p->end->next=p1;
p->end=p1;
}
p->size++;
}
char* top(struct que p)
{
return p.beg->info;
}
int size(const struct que* p)
{
return p->size;
}
int empty(struct que p)
{
if (p.size>0) return 0; else return 1;
}
void ini_que(struct que* p, int n)
{
p->beg=NULL;
p->end=NULL;
p->size=0;
p->width=n;
}
int main()
{
```

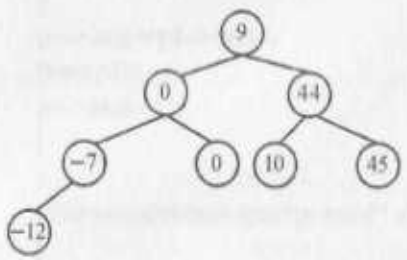
```
int i,m;
int*pi;
struct que ps;
ini_que(&ps, sizeof(int));
for(i=0; i<5; i++) push(&ps, &i);
while(!empty(ps))
{
pi=(int*)top(ps);
printf("%d", *pi);
pop(&ps);
}
getch();
return 0;
}
```

10.3. Universal daraxt

Daraxt bu ajdod-avlod munosabatlari bilan bog'langan tugun deb ataluvchi elementlar ierarxik strukturasidir. Hech qanday ajdodga ega bo'lmagan yagona tugun ildiz deb ataladi. Hech qanday avlodga ega bo'lmagan tugunlar barglar deyiladi.

Ikkilik binar daraxtda har bir tugun bir yoki ikki farzand tugun bilan bog'liq bo'ladi. Rekursiv ravishda binar daraxt quyidagicha aniqlanadi: agar binar daraxt bo'sh bo'lmasa, bitta ildiz va chap hamda o'ng ostki daraxtga ega bo'ladi.

Binar izlash daraxti bo'sh bo'lmasa, quyidagi xossaga ega: har bir tugun shu tugun ildiz bo'lgan chap ostki daraxt hamma elementlaridan katta va o'ng ostki daraxt hamma elementlaridan kichik. Bunday daraxtlarda izlash tezligi: $O(n)=C\cdot\log_2 n$ (eng yomon holda $O(n)=n$).



Misol. Quyidagi 9, 44, 0, –7, 10, 6, –12, 45 sonlar uchun binar izlash daraxti

Binar daraxtlar uchun quyidagi amallar aniqlangan:

- element qo'shish;
- element o'chirish;
- daraxtni aylanib chiqish;

- izlash.

Universal navbat har bir tuguni axborot qismi void turidagi ko'rsatkichdan iborat struturadir:

```
struct sbtree_node
{
void* info;
struct sbtree_node* lchild;
struct sbtree_node* rchild;
};
```

Bu yerda lchild chap farzandga ko'rsatkich va rchild o'ng farzandga ko'rsatkich.

Daraxt o'zi alohida struktura sifatida kiritilgan:

```
struct sbtree
{
struct sbtree_node* beg;
int size;
int width;
};
```

Bu yerda beg ildizga ko'rgatkich, width ma'lumot hajmi, size navbatdagi elementlar soni.

Asosiy funksiyalar:

char* creat_sbtree_node(void*val, int width) – yangi tugun yaratish yordamchi funksiyasi. Bu yerda val kiritilayotgan ma'lumotga ko'rsatkich va width ma'lumot hajmi;

void insert(struct sbtree*p, void* val, int fmp(void*p1, void*p2)) – yangi elementni daraxtga qo'shish. Bu yerda val kiritilayotgan ma'lumotga ko'rsatkich va fmp elementlar qiymatlarini solishtirish funksiyasiga ko'rsatkich;

int find(struct sbtree*p, void* val, int fmp(void*p1, void*p2)) – elementni topish;

void round_sbtree(struct sbtree_node *p, void pmp(void*val)) – hamma elementlarni aylanib chiqish rekursiv funksiyasi. Har bir elementga pmp funksiyasi qo'llanadi;

int empty(struct que p) – navbat bo'shligini tekshirish;

int size (struct que p) – navbat elementlari soni.

Bundan tashqari daraxtni initsiallash uchun quyidagi sarlavhali funksiya kiritilgan:

void ini_sbtree(struct sbtree* p, int n) – bu yerda n kiritilayotgan ma'lumotlar hajmi.

Dastur matni:

```
#include<stdio.h>
#include<stdlib.h>
#include<mem.h>
#include<conio.h>
struct sbtree_node
{
void* info;
struct sbtree_node* lchild;
struct sbtree_node* rchild;
};

struct sbtree
{
struct sbtree_node* beg;
int size;
int width;
};

char* creat_sbtree_node(void*val, int width)
{
struct sbtree_node* pn;
pn=(struct sbtree_node*)malloc(sizeof(struct sbtree_node));
pn->info=malloc(width);
memcpy(pn->info, val, width);
pn->lchild=NULL;
pn->rchild=NULL;
return (char*)pn;
};
void insert(struct sbtree*p, void* val, int fmp(void*p1, void*p2))
{
```



```
char c=' ';
struct sbtree_node* p1;
struct sbtree_node* p2;
if(p->size==0)
{
p->beg=(void*)creat_sbtree_node(val, p->width);
p->size=1;
return;
}

p1=p->beg;
while(p1!=NULL)
{
if (fmp(p1->info, val)==0) return;

p2=p1;
if(fmp(p1->info, val)>0)
{
c='l';
p1=p1->lchild;
continue;
}

if(fmp(p1->info, val)<0)
{
c='r';
p1=p1->rchild;
continue;
}
}
p1=(void*)creat_sbtree_node(val, p->width);
if (c == 'l') p2->lchild=p1; else p2->rchild=p1;
p->size++;
}
int find(struct sbtree*p, void* val, int fmp(void*p1, void*p2))
{
```

```
struct sbtree_node* p1;
if(p->size == 0) return 0;
p1=p->beg;
while(p1!=NULL)
{
if (fmp(p1->info, val) == 0) return 1;
if(fmp(p1->info, val)>0) p1=p1->lchild;
if(fmp(p1->info, val)<0) p1=p1->rchild;
}
return 0;
}
void round_sbtree(struct sbtree_node *p, void pmp(void*val))
{
if(p!=NULL)
{
pmp(p->info);
round_sbtree(p->lchild, pmp);
round_sbtree(p->rchild, pmp);
}
}

int size(const struct sbtree* p)
{
return p->size;
}

int empty(struct sbtree p)
{
if (p.size>0) return 0; else return 1;
}

void ini_sbtree(struct sbtree* p, int n)
{
p->beg=NULL;
p->size=0;
```



```
p->width=n;
}

int smp(void*a, void*b)
{
int* pa=(int*)a;
int* pb=(int*)b;

if((*pa)<(*pb)) return 1;
if((*pa)>(*pb)) return -1;
if((*pa) == (*pb)) return 0;

return 1;
}
void pmp(void* val)
{
int* pa=val;
printf("%d", *pa);
}
int main()
{
int i, k,m;
int a[]={3, 15, 2, 4, 15};
struct sbtree sp;
ini_sbtree(&sp, sizeof(int));
for(i=0; i<5; i++)
insert(&sp, &a[i], smp);
round_sbtree(sp.beg, pmp);
printf("\n \n");
for(i=0; i<5; i++)
{
k=find(&sp, &i, smp);
printf("%d\n", k);
}
getch();
return 0;
}
```

10-bob bo'yicha savollar

1. Dinamik informatsion strukturalarga ta'rif bering.
2. Universal struktura deb qanday strukturaga aytiladi?
3. Ro'yxat deb qanday strukturaga aytiladi?
4. Ro'yxatlar turlarini ko'rsating.
5. Stek qanday prinsipda ishlaydi?
6. Navbat deb qanday strukturaga aytiladi?
7. Daraxt qanday struktura?
8. Binar daraxt deb qanday strukturaga aytiladi?
9. Izlash daraxti xususiyatlari.
10. Daraxtda sikl mavjud bo'lishi mumkinmi?

10-bob bo'yicha topshiriqlar

1. Ikki bo'g'inli navbat strukturasini yarating.
2. Ikki tomonli navbat strukturasini yarating.
3. Izlash daraxtidan element o'chirish funksiyasini tuzing.
4. Ikki izlash daraxtini qo'shib, uchinchi izlash daraxtini hosil qiluvchi funksiya tuzing.
5. Prioritetli navbat strukturasini binar daraxt asosida yarating. Prioritetli navbat shunday navbatki, eng kichik element o'qiladi va o'chiriladi.

II QISM. C++ TILIDA OBYEKTLI DASTUSLASH

11-bob. OBYEKTGA MO'LJALLANGAN DASTURLASH ASOSLARI

11.1. Obyektga mo'ljallangan yondashuv tarixi

Obyektga mo'ljallangan yondashuv (OMY) dasturiy ta'minotning tabiiy rivojidagi navbatdagi pog'onadir. Vaqt o'tishi bilan qaysi uslublar ishlash uchun qulay-u, qaysinisi noqulay ekanini aniqlash oson bo'lib bordi. OMY eng muvaffaqiyatli, vaqt sinovidan o'tgan uslublarni o'zida samarali mujassam etadi.

Dastlab dasturlar kommutatsiya bloki orqali kompyuterning asosiy xotirasiga to'g'ridan to'g'ri kiritilar edi. Dasturlar mashina tillarida ikkilik tasavvurda yozilar edi. Dasturlarni mashina tilida yozishda tez-tez xatolarga yo'l qo'yilar edi, buning ustiga ularni tuzilmalashtirishning imkoni bo'lmagani tufayli, kodni kuzatib borish amalda deyarli mumkin bo'lmagan hol edi. Bundan tashqari, mashina kodlaridagi dasturni tushunish g'oyat murakkab edi.

Vaqt o'tishi bilan kompyuterlar tobora kengroq qo'llana boshlandi hamda yuqoriroq darajadagi protsedura tillari paydo bo'ldi. Bularning dastlabkisi FORTRAN tili edi. Biroq OMYning rivojiga asosiy ta'sirni keyinroq paydo bo'lgan, masalan, ALGOL kabi protsedura tillari ko'rsatdi.

Protseduraviy yondoshuv. Shu vaqtgacha dasturlar berilgan ma'lumotlar ustida biror-bir amal bajaruvchi protseduralar ketma-ketligidan iborat edi. Protsedura yoki funksiya ham o'zida aniqlangan ketma-ket bajariluvchi komandalar to'plamidan iborat. Bunda berilgan ma'lumotlarga murojaatlar protseduralarga ajratilgan holda amalga oshiriladi.

Protsedura tillari dasturchiga axborotga ishlov berish dasturini pastroq darajadagi bir nechta protseduraga bo'lib tashlash imkonini beradi. Pastroq darajadagi bunday protseduralar dasturning umumiy tuzilmasini belgilab beradi. Ushbu protseduralarga izchil murojaatlar protseduralardan tashkil topgan dasturlarning bajarilishini boshqaradi.

Dasturlashning bu yangi paradigmasi mashina tilida dasturlash paradigmasiga nisbatan ancha ilg'or bo'lib, unga tuzilmalashtirishning asosiy vositasi bo'lgan protseduralar qo'shilgan edi. Kichik funksiyalarni nafaqat tushunish, balki sozlash ham osonroq kechadi.

Strukturaviy dasturlashning asosiy g'oyasi «bo'lakla va hukmronlik qil» prinsipiga butunlay mos keladi. Kompyuter dasturini masalalar to'plamidan iborat deb qaraymiz. Oddiy tavsiflash uchun murakkab bo'lgan ixtiyoriy masalani bir nechta, nisbatan kichikroq bo'lgan, tarkibiy masalalarga ajratamiz va bo'linishni toki masalalar tushunishi uchun yetarli darajada oddiy bo'lguncha davom ettiramiz.

Misol sifatida kompaniya xizmatchilarining o'rtacha ish haqini hisoblashni olamiz. Bu masala sodda emas. Uni qator qism masalalarga bo'lamiz:

1. Har bir xizmatchining oylik maoshi qanchaligini aniqlaymiz.
2. Kompaniya xodimlari sonini aniqlaymiz.
3. Barcha ish haqlarini yig'amiz.
4. Hosil bo'lgan yig'indini kompaniya xodimlari soniga bo'lamiz.

Xodimlarning oylik maoshlari yig'indisini hisoblash jarayonini ham bir necha bosqichlarga ajratish mumkin.

1. Har bir xodim haqidagi yozuvni o'qiymiz.
2. Ish haqi to'g'risidagi ma'lumotni olamiz.
3. Ish haqi qiymatini yig'indiga qo'shamiz.
4. Keyingi xodim haqidagi yozuvni o'qiymiz.

O'z navbatida, har bir xodim haqidagi yozuvni o'qish jarayonini ham nisbatan kichikroq qism operatsiyalarga ajratish mumkin:

1. Xizmatchi faylini ochamiz.
2. Kerakli yozuvga o'tamiz.
3. Ma'lumotlarni diskdan o'qiymiz.

Strukturaviy dasturlash murakkab masalalarni yechishda yetarlicha muvafaqqiyatli uslub bo'lib qoldi. Lekin 1980-yillar oxirlarida strukturaviy dasturlashning ham ayrim kamchiliklari ko'zga tashlandi.

Birinchidan, berilgan ma'lumotlar (masalan, xodimlar haqidagi yozuv) va ular ustidagi amallar (izlash, tahrirlash) bajarilishining bir butun tarzda tashkil etilishidek tabiiy jarayon realizatsiya qilinmagan edi. Aksincha, protseduraviy dasturlash berilganlar strukturasini bu ma'lumotlar ustida amallar bajaradigan funksiyalarga ajratgan edi.

Ikkinchidan, dasturchilar doimiy tarzda eski muammolarning yangi yechimlarini ixtiro qilar edilar. Bu vaziyat ko'pincha velosipedni qayta ixtiro

qilish ham deb aytiladi. Ko'plab dasturlarda takrorlanuvchi bloklarni ko'p martalab qo'llash imkoniyatiga bo'lgan xohish tabiiydir. Buni radio ishlab chiqaruvchi tomonidan priyomnikni yig'ishga o'xshatish mumkin. Konstruktor har safar diod va tranzistorni ixtiro qilmaydi. U oddiygina – oldin tayyorlangan radio detallaridan foydalanadi, xolos. Dasturiy ta'minotni ishlab chiquvchilar uchun esa bunday imkoniyat ko'p yillar mobaynida yo'q edi.

Boshqa tomondan, protsedurali dasturlash koddan takroran foydalanish imkonini cheklab qo'yadi. Va, nihoyat, shu narsa aniq bo'ldiki, protsedurali dasturlash usullari bilan dasturlarni ishlab chiqishda diqqatni ma'lumotlarga qaratishning o'zi muammolarni keltirib chiqarar ekan. Chunki ma'lumotlar va protsedura ajralgan, ma'lumotlar inkapsulatsiyalanmagan. Bu nimaga olib keladi? Shunga olib keladiki, har bir protsedura ma'lumotlarni nima qilish kerakligini va ular qayerda joylashganini bilmog'i lozim bo'ladi. Agar protsedura ma'lumotlar ustidan noto'g'ri amallarni bajarsa, u ma'lumotlarni buzib qo'yishi mumkin. Har bir protsedura ma'lumotlarga kirish usullarini dasturlashi lozim bo'lganligi tufayli, ma'lumotlar taqdimotining o'zgarishi dasturning ushbu kirish amalga oshirilayotgan barcha o'rinlarining o'zgarishiga olib kelar edi. Shunday qilib, hatto eng kichik to'g'rilash ham butun dasturda qator o'zgarishlarni sodir bo'lishiga olib kelar edi.

Modulli dasturlashda, masalan, Modula2 kabi tilda protsedurali dasturlashda topilgan ayrim kamchiliklarni bartaraf etishga urinib ko'rildi. Modulli dasturlash dasturni bir necha tarkibiy bo'laklarga, yoki, boshqacha qilib aytganda, modullarga bo'lib tashlaydi. Agar protsedurali dasturlash ma'lumotlar va jarayonlarni bo'lib tashlasa, modulli dasturlash, undan farqli o'laroq, ularni birlashtiradi. Modul ma'lumotlarning o'zidan hamda ma'lumotlarga ishlov beradigan protseduralardan iborat. Dasturning boshqa qismlariga moduldan foydalanish kerak bo'lib qolsa, ular modul interfeysiga murojaat etadi. Modullar barcha ichki axborotni dasturning boshqa qismlarida yashiradi.

Biroq modulli dasturlash ham kamchiliklardan xoli emas. Modullar kengaymas bo'ladi, bu degani kodga bevosita kirishsiz hamda uni to'g'ridan to'g'ri o'zgartirmay turib modulni qadam-baqadam o'zgartirish mumkin emas. Bundan tashqari, bitta modulni ishlab chiqishda, uning funksiyalarini boshqasiga o'tkazmay (delegat qilmay) turib boshqasidan foydalanib bo'lmaydi. Yana garchi modulda turni belgilab bo'lsa-da, bir modul boshqasida belgilangan turdan foydalana olmaydi.

Modulli va prosedurali dasturlash tillarida turni kengaytirish usuli, agar «agregatlash» deb ataluvchi usul yordamida boshqa turlarni yaratishni hisobga olmaganda, mavjud emas.

Va, nihoyat, modulli dasturlash – bu yana protseduraga mo'ljallangan gibridli sxema bo'lib, unga amal qilishda dastur bir necha protseduralarga bo'linadi. Biroq endilikda protseduralar ishlov berilmagan ma'lumotlar ustida amallarni bajarmaydi, balki modullarni boshqaradi.

Amaliyotga dostona foydalanuvchi interfeyslari, ramkali oyna, menu va ekranlarning tadbiq etilishi dasturlashda yangi uslubni keltirib chiqardi. Dasturlarni ketma-ket boshidan oxirigacha emas, balki uning alohida bloklari bajarilishi talab qilinadigan bo'ldi. Biror-bir aniqlangan hodisa yuz berganda dastur unga mos shaklda ta'sir ko'rsatishi lozim. Masalan, bir tugma bosilganda faqatgina unga biriktirilgan amallar bajariladi. Bunday uslubda dasturlar ancha interaktiv bo'lishi lozim. Buni ularni ishlab chiqishda hisobga olish lozim.

Obyektga mo'ljallangan dasturlash (OMD) bu talablarga to'la javob beradi. Bunda dasturiy komponentlarni ko'p martalab qo'llash va berilganlarni manipulatsiya qiluvchi usullar bilan birlashtirish imkoniyati mavjud.

Obyektga mo'ljallangan dasturlashning asosiy maqsadi berilganlar va ular ustida amal bajaruvchi protseduralarni yagona obyekt deb qarashdan iboratdir.

11.2. Obyektga mo'ljallangan yondashuvning afzalliklari va maqsadlari

OMY dasturiy ta'minotni ishlab chiqishda oltida asosiy maqsadni ko'zlaydi. OMY paradigmasiga muvofiq ishlab chiqilgan dasturiy ta'minot quyidagi xususiyatlarga ega bo'lmog'i lozim:

1) tabiiylik;
2) ishonchlilik;
3) qayta qo'llanish imkoniyati;
4) kuzatib borishda qulaylik;
5) takomillashishga qodirlik;
6) yangi versiyalarni davriy chiqarishning qulayligi.

Tabiiylik. OMY yordamida tabiiy dasturiy ta'minot yaratiladi. Tabiiy dasturlar tushunarliroq bo'ladi. Dasturlashda «massiv» yoki «xotira sohasi» kabi atamalardan foydalanish o'rniga, yechilayotgan masala mansub bo'lgan soha atamalaridan foydalanish mumkin. Ishlab chiqilayotgan

dasturni kompyuter tiliga moslash o'rniga, OMY aniq bir sohaning atamalaridan foydalanish imkonini beradi.

Ishonchlilik. Yaxshi dasturiy ta'minot boshqa har qanday mahsulotlar, masalan, muzlatgich yoki televizorlar kabi ishonchli bo'lmog'i lozim.

Puxta ishlab chiqilgan va tartib bilan yozilgan obyektga mo'ljallangan dastur ishonchli bo'ladi. Obyektlarning modulli tabiati dastur qismlaridan birida, uning boshqa qismlariga tegmagan holda, o'zgartirishlarni amalga oshirish imkonini beradi. Obyekt tushunchasi tufayli, axborotga ushbu axborot kerak bo'lgan shaxslar egalik qiladi, ma'suliyat esa berilgan funksiyalarni bajaruvchilar zimmasiga yuklatiladi.

Qayta qo'llanish imkoniyati. Quruvchi uy qurishga kirishar ekan, har gal g'ishtlarning yangi turini ixtiro qilmaydi. Radiomuhandis yangi sxemani yaratishda, har gal rezistorlarning yangi turini o'ylab topmaydi. Unda nima uchun dasturchi «G'ildirak ixtiro qilaverishi kerak»? Masala o'z yechimini topgan ekan, bu yechimdan ko'p martalab foydalanish lozim.

Malakali ishlab chiqilgan obyektga mo'ljallangan sinflarni bemalol takroran ishlatish mumkin. Xuddi modullar kabi, obyektlarni ham turli dasturlarda takroran qo'llash mumkin. Modulli dasturlashdan farqli o'laroq, OMY mavjud obyektlarni kengaytirish uchun vorislikdan, sozlanayotgan kodni yozish uchun esa polimorfizmdan foydalanish imkonini beradi.

Kuzatib borishda qulaylik. Dasturiy mahsulotning ish berish davri uning ishlab chiqilishi bilan tugamaydi. Dasturni ishlatish jarayonida *kuzatib borish* deb nomlanuvchi jarayon muhimdir. Dasturga sarflangan 60 foizdan 80 foizgacha vaqt kuzatib borishga ketadi. Ishlab chiqish esa ish berish siklining 20 foizinigina tashkil etadi.

Puxta ishlangan obyektga mo'ljallangan dastur ishlatishda qulay bo'ladi. Xatoni bartaraf etish uchun faqat bitta o'ringa to'g'rilash kiritish kifoya qiladi. Chunki ishlatishdagi o'zgarishlar tiniq, boshqa barcha obyektlar takomillashtirish afzalliklaridan avtomatik ravishda foydalana boshlaydi. O'zining tabiiyligi tufayli dastur matni boshqa ishlab chiquvchilar uchun tushunarli bo'lmog'i lozim.

Kengayishga qodirlik. Foydalanuvchilar dasturni kuzatib borish paytida tez-tez tizimga yangi funksiyalarni qo'shishni iltimos qiladilar. Obyektlar kutubxonasini tuzishning o'zida ham ushbu obyektlarning funksiyalarini kengaytirishga to'g'ri keladi.

Dasturiy ta'minot statik (qotib qolgan) emas. Dasturiy ta'minot foydali bo'lib qolishi uchun, uning imkoniyatlarini muttasil kengaytirib borish

lozim. OMY da dasturni kengaytirish usullari ko'p. Vorislik, polimorfizm, qayta aniqlash, vakillik hamda ishlab chiqish jarayonida foydalanish mumkin bo'lgan ko'plab boshqa shablonlar shular jumlasidandir.

Yangi versiyalarning davriy chiqarilishi. Zamonaviy dasturiy mahsulotning ish berish davri ko'p hollarda haftalar bilan o'lchanadi. OMY tufayli dasturlarni ishlab chiqish davrini qisqartirishga erishildi, chunki dasturlar ancha ishonchli bo'lib bormoqda, kengayishi osonroq hamda takroran qo'llanishi mumkin.

Dasturiy ta'minotning tabiiyligi murakkab tizimlarning ishlab chiqilishini osonlashtiradi. Har qanday ishlanma hafsala bilan yondashuvni talab qiladi, shuning uchun tabiiylik dasturiy ta'minotning ishlab chiqish davrlarini qisqartirish imkonini beradi, chunki butun diqqat-e'tiborni yechilayotgan masalaga jalb qildiradi.

Dastur qator obyektlarga bo'lingach, har bir alohida dastur qismini boshqalari bilan parallel ravishda ishlab chiqish mumkin bo'ladi. Bir nechta ishlab chiquvchi sinflarni bir-birlaridan mustaqil ravishda ishlab chiqishi mumkin bo'ladi. Ishlab chiqishdagi bunday parallellik ishlab chiqish vaqtini qisqartiradi.

11.3. Inkapsulatsiya tushunchasi

C++ tili obyektga OMD quyidagi prinsiplariga tayanadi:

- inkapsulatsiya;
- merosxo'rlik;
- polimorfizm.

Inkapsulatsiya. Inkapsulatsiyalash – ma'lumotlarning va shu ma'lumotlar ustida ish olib boradigan kodlarning bitta obyektda birlashtirilishi. OMD atamachiligida ma'lumotlar obyekt ma'lumotlari a'zolari (data members) deb, kodlar obyektli usullar yoki funksiya-a'zolari (methods, member functions) deb ataladi.

Inkapsulatsiya yordamida berilganlarni yashirish ta'minlanadi. Bu juda yaxshi xarakteristika bo'lib, foydalanuvchi o'zi ishlatayotgan obyektning ichki ishlari haqida umuman o'ylamaydi. Haqiqatan ham, sovitgichni ishlatishda refrijektorning ishlash prinsipini bilish shart emas. Yaxshi ishlab chiqilgan dastur obyektini qo'llashda uning ichki o'zgaruvchilarining o'zaro munosabati haqida qayg'urish zarur emas.

C++ tilida inkapsulatsiya prinsipi sinf deb ataluvchi nostandart turlarni (foydalanuvchi turlarini) hosil qilish orqali himoya qilinadi.

Sinf – bu maxsus tur bo'lib, o'zida maydon, usullar va xossalarni mujassamlashtiradi.

Sinf murakkab struktura bo'lib, ma'lumotlar ta'riflaridan tashqari, protsedura va funksiyalar ta'riflarini o'z ichiga oladi.

Sinf jismoniy mohiyatga ega emas, tuzilmaning e'lon qilinishi uning eng yaqin analogidir. Sinf obyektni yaratish uchun qo'llangandagina, xotira ajralib chiqadi. Bu jarayon ham sinf nusxasi (class intsance)ni yaratish deb ataladi.

To'g'ri aniqlangan sinf obyektini butun dasturiy modul sifatida ishlatish mumkin. Haqiqiy sinfning barcha ichki ishlari yashirin bo'lishi lozim. To'g'ri aniqlangan sinfning foydalanuvchilari uning qanday ishlashini bilishi shart emas, ular sinf qanday vazifani bajarishini bilsalar yetarli.

Aynan inkapsulatsiyalash tufayli mustaqillik darajasi ortadi, chunki ichki detallar interfeys ortida yashiringan bo'ladi.

Inkapsulatsiyalash modullikning obyektga mo'ljallangan tavsifidir. Inkapsulatsiyalash yordamida dasturiy ta'minotni ma'lum funksiyalarni bajaruvchi modullarga bo'lib tashlash mumkin. Bu funksiyalarni amalga oshirish detallari esa tashqi olamdan yashirin holda bo'ladi.

Mohiyatan *inkapsulatsiyalash* atamasi «germetik berkitilgan; tashqi ta'sirlardan himoyalangan dastur qismi» degan ma'noni bildiradi.

Agar biror-bir dasturiy obyektga inkapsulatsiyalash qo'llanilgan bo'lsa, u holda bu obyekt qora quti sifatida olib qaraladi. Siz qora quti nima qilayotganini uning tashqi interfeysini ko'rib turganingiz uchungina bilishingiz mumkin. Qora quti biror narsa qilishga majburlash uchun, unga xabar yuborish kerak. Qora quti ichida nima sodir bo'layotgani ahamiyatli emas, qora quti yuborilgan xabarga adekvat (mos ravishda) munosabatda bo'lishi muhimroqdir.

Interfeys tashqi olam bilan tuzilgan o'ziga xos bitim bo'lib, unda tashqi obyektlar ushbu obyektga qanday talablar yuborishi mumkinligi ko'rsatilgan bo'ladi. Interfeys – obyektni boshqarish pulti.

Shuni ta'kidlab o'tish lozimki, «Casio» soatining suyuq kristalli displeyi ushbu obyektning ma'lumotlar a'zosi bo'ladi, boshqarish tugmachalari esa obyektli usullar bo'ladi. Soat tugmachalarini bosib, displeyda vaqtni o'rnatish ishlarini olib borish mumkin, ya'ni OMD atamalarini qo'llaydigan bo'lsak, usullar, ma'lumotlar a'zolarini o'zgartirib, obyekt holatini modifikasiya qiladi.

Agarda muhandis ishlab chiqarish jarayonida rezistorni qo'llasa, u buni yangidan ixtiro qilmaydi, omborga (do'konqa) borib mos para-

metrlarga muvofiq kerakli detalni tanlaydi. Bu holda muhandis joriy rezistor qanday tuzilganligiga e'tiborini qaratmaydi, rezistor faqatgina zavod xarakteristikalariga muvofiq ishlasa yetarlidir. Aynan shu tashqi konstruksiyada qo'llaniladigan yashirinlik yoki obyektni yashirinligi yoki avtonomligi xossasi inkapsulatsiya deyiladi.

Yana bir marta takrorlash joizki, rezistorni samarali qo'llash uchun uning ishlash prinsipi va ichki qurilmalari haqidagi ma'lumotlarni bilish umuman shart emas. Rezistorning barcha xususiyatlari inkapsulatsiya qilingan, ya'ni yashirilgan. Rezistorning faqatgina o'z funksiyasini bajarishi yetarlidir.

Inkapsulatsiyalash maqsadi. Inkapsulatsiyalashdan to'g'ri foydalanish tufayli obyektlar bilan o'zgartiriladigan komponentlar (tarkibiy qismlar)dek muomala qilish mumkin. Bir obyekt boshqa obyektdan foydalana olishi uchun, u birinchi obyektning ommaviy interfeysidan qanday fodalanish kerakligini bilishi kifoya. Bunday mustaqillik uchta muhim afzallikka ega.

1. Mustaqillik tufayli obyektdan takroran foydalanish mumkin. Inkapsulatsiyalash puxta amalga oshirilgan bo'lsa, obyektlar ma'lum bir dasturga bog'lanib qolgan bo'lmaydi. Ulardan imkoni bo'lgan hamma yerda foydalanish mumkin bo'ladi. Obyektdan boshqa biror o'rinda foydalanish uchun, uning interfeysidan foydalanib qo'ya qolish kifoya.

2. Inkapsulatsiyalash tufayli, obyektda boshqa obyektlar uchun ko'rinmas bo'lgan o'zgarishlarni amalga oshirish mumkin. Agar interfeys o'zgartirilmasa, barcha o'zgarishlar obyektdan foydalanayotganlar uchun ko'rinmas bo'ladi. Inkapsulatsiyalash komponentni yaxshilash, amalga oshirish samaradorligini ta'minlash, xatolarni bartaraf etish imkonini beradi hamda bularning hammasi dasturning boshqa obyektlariga ta'sir ko'rsatmaydi. Obyektdan foydalanuvchilar ularda amalga oshirilayotgan barcha o'zgarishlardan avtomatik tarzda o'tadilar.

3. Himoyalangan obyektdan foydalanishda obyekt va dasturning boshqa qismi o'rtacida biror-bir ko'zda tutilmagan o'zaro aloqalar bo'lishi mumkin emas. Agar obyekt boshqalardan ajratilgan bo'lsa, bu holda u dasturning boshqa qismi bilan faqat o'z interfeysi orqali aloqaga kirishishi mumkin.

Shunday qilib, inkapsulatsiyalash yordamida modulli dasturlarni yaratish mumkin.

11.4. Samarali inkapsulatsiyalash

Samarali inkapsulatsiyalashning uchta o'ziga xos belgisi quyidagicha:

– abstraksiya;

– joriy qilishning berkitilganligi;

– ma'suliyatning bo'linganligi.

Abstraksiya. Garchi obyektga mo'ljallangan tillar inkapsulatsiyalashdan foydalanishga yordam bersa-da, biroq ular inkapsulatsiyalashni kafolatlamaydi. Tobe va ishonchsiz kodni yaratib qo'yish oson. Samarali inkapsulatsiyalash – sinchkovlik bilan ishlab chiqish hamda abstraksiya va tajribadan foydalanish natijasi. Inkapsulatsiyalashdan samarali foydalanish uchun dasturni ishlab chiqishda avval abstraksiyadan va uning bilan bog'liq konsepsiyalardan foydalanishni o'rganib olish lozim.

Abstraksiya murakkab masalani soddalashtirish jarayonidir. Muayyan masalani yechishga kirishar ekansiz, siz barcha detallarni hisobga olishga urinmaysiz, balki yechimni osonlashtiradiganlarini tanlab olasiz.

Aytaylik, siz yo'l harakati modelini tuzishingiz kerak. Shunisi ayonki, bu o'rinda siz svetoforlar, mashinalar, shosselar, bir tomonlama va ikki tomonlama ko'chalar, ob-havo sharoitlari va h.k. sinflarini yaratasiz. Ushbu elementlarning har biri transport harakatiga ta'sir ko'rsatadi. Biroq bu o'rinda hasharotlar va qushlar ham yo'lda paydo bo'lishi mumkin bo'lsa-da, siz ularning modelini yaratmaysiz. Chunki, siz mashinalar markalarini ham ajratib ko'rsatmaysiz. Siz haqiqiy olamni soddalashtirasiz hamda uning faqat asosiy elementlaridan foydalanasiz. Mashina – modelning muhim detali, biroq bu Kadillakmi yoki boshqa biror markadagi mashinami, yo'l harakati modeli uchun bu detallar ortiqcha.

Abstraksiyaning ikkita afzal jihati bor. Birinchidan, u masala yechimini soddalashtiradi. Muhimi shundaki, abstraksiya tufayli dasturiy ta'minot komponentlaridan takroran foydalanish mumkin. Takroran qo'llanadigan komponentlarni yaratishda ular, odatda, g'oyat ixtisoslashadi. Ya'ni komponentlar biror-bir ma'lum masala yechimiga mo'ljallangani, yana ular keraksiz o'zaro bog'liqlikda bo'lgani sababli dastur fragmentining boshqa biror o'rinda takroran qo'llanishi qiyinlashadi. Imkoni boricha bir qator masalalarni yechishga qaratilgan obyektlarni yaratishga harakat qiling. Abstraksiya bitta masala yechimidan ushbu sohadagi boshqa masalalarni ham yechishda foydalanish imkonini beradi.

Ikkita misolni ko'rib chiqamiz.

Birinchi misol: bank kassasiga navbatda turgan odamlarni tasavvur qiling. Kassir bo'shaganda, uning darchasiga navbatda turgan birinchi mijoz yaqinlashadi. Shunday qilib, navbatdagi hamma odam birin-ketin kassir darchasi tomon suriladi. Navbatda turganlar «birinchi kelganga birinchi bo'lib xizmat ko'rsatildi» algoritmi bo'yicha surilib boradi.

Ikkinchi misol: gazakxonada gamburgerli konveyerni ko'rib chiqaylik. Navbatdagi yangi gamburger konveyerga kelib tushganda, u gamburgerlar qatoridagi oxirgi gamburger yonidan joy oladi. Shuning uchun konveyerdan olingan gamburger u yerda boshqalaridan ko'proq vaqt turib qolgan bo'ladi. Restoranlar «birinchi kelganga birinchi bo'lib xizmat ko'rsatildi» algoritmi bo'yicha ishlaydi.

Garchi bu misollar butkul turlicha bo'lsa-da, ularda qandaydir umumiy tamoyil qo'llangan bo'lib, undan boshqa vaziyatlarda ham foydalanish mumkin. Boshqacha qilib aytganda, siz abstraksiyaga kelasiz.

Bu misollarning har ikkalasida ham «birinchi kelganga birinchi bo'lib xizmat ko'rsatildi» algoritmi qo'llanilgan. Bu o'rinda navbat elementi nimani bildirishi muhim emas. Haqiqatda ushbu element navbat oxiriga kelib qo'shilishi hamda navbatni uning boshiga yetganda tark etishigina muhimdir.

Abstraksiya yordamida bir marta navbatni yaratib, keyinchalik uni boshqa dasturlarni yozishda qo'llash mumkinki, bu dasturlarda elementlarga «birinchi kelganga birinchi bo'lib xizmat ko'rsatildi» algoritmi bo'yicha ishlov beriladi.

Samarali abstraksiyani bajarish uchun bir nechta qoidalarni ifodalash mumkin.

– Qandaydir aniq holatni emas, umumiy holatni olib qarang.

– Turli masalalarga xos bo'lgan umumiy jihatni izlab toping. Shunchaki alohida xodisani emas, asosiy tamoyilni ko'ra bilishga harakat qiling.

– Garchi abstraksiya g'oyat qimmatli bo'lsa-da, biroq eng yechimli masalani yodingizdan chiqarmang.

– Abstraksiya hamma vaqt ham ochiq-oydin emas. Masalani yechar ekansiz, siz birinchi, ikkinchi va hatto, uchinchi martasiga ham abstraksiyani tanib ololmasligingiz mumkin.

– Muvaffaqiyatsizlikka tayyor turing. Amalda har bir vaziyat uchun to'g'ri keladigan abstrakt dasturni yozish mumkin emas.

Abstraksiyani so'nggi maqsad sifatida emas, balki unga erishish yo'lidagi vosita sifatida olib qarash kerak. Muayyan hollarda abstraksiyani qo'llash kerak emas. Yaxshigina evristik qoida mavjud bo'lib, unga ko'ra, agar siz biror-bir masalani o'zaro o'xshash usullar bilan kamida uch marta yechgan bo'lsangiz, abstraksiyani faqat shunday masalalarga qo'llash tavsiya qilinadi.

Abstrakt komponentni takroran qo'llash osonroq, chunki u biror-bir bitta o'ziga xos masalani yechishga emas, balki qator masalalarni

yechishga mo'ljallangan. Biroq bu hol komponentdan shunchaki takroran foydalanishdan ko'ra ko'proq inkapsulatsiyalashga tegishli. Ichki detallarni yashirishga o'rganish g'oyat muhim. Ma'lumotlarning abstrakt turlarini qo'llash inkapsulatsiyalashni samarali qo'llashga imkon beradi.

Joriy qilishni yashirish yordamida sirlarni yashirish. Abstraksiya samarali inkapsulatsiyalashning tarkibiy qismlaridan biri xolos. Tashqi ta'sirlardan mutlaqo himoyalanmagan abstrakt dasturni ham yozish mumkin. Aynan shuning uchun obyektning ichki joriy qilinishini berkitish kerak bo'ladi.

Joriy qilishning berkitilganligi. Joriy qilishning berkitilganligi ikkita afzallikka ega:

– obyektlarni foydalanuvchilardan himoyalaydi;

– foydalanuvchilarni obyektlardan himoyalaydi.

Birinchi afzallik – obyektlarni himoyalashni ko'rib chiqamiz.

Asl inkapsulatsiyalash til darajasida qurilma til konstruksiyalari yordamida ta'minlanadi.

Ma'lumotlarning abstrakt turlari – bu ma'lumotlar va ular ustida o'tkaziladigan operatsiyalar to'plami.

Ma'lumotlarning abstrakt turlari ichki axborot va holatni puxta ishlab chiqilgan interfeys ortida yashirar ekan, ular tilda ma'lumotlarning yangi turlarini aniqlashga imkon beradi. Bunday interfeysda ma'lumotlarning abstrakt turlari bo'linmas butunlik sifatida taqdim etilgan. Ma'lumotlarning abstrakt turlari inkapsulatsiyalashni qo'llashni osonlashtiradi, chunki ular tufayli inkapsulatsiyalashni vorisliksiz va polimorfizmsiz qo'llash mumkin, bu esa inkapsulatsiyalashning aynan o'ziga diqqatni qaratish imkonini beradi. Ma'lumotlarning abstrakt turlari, shuningdek, tur tushunchasining qo'llanishini ham osonlashtiradi. Agar tur nima ekanini anglab olsak, bu holda OMY ixtisoslashtirilgan foydalanuvchilik turlari yordamida tilni kengaytirishning tabiiy usulini taklif qilayotganini oson sezib olish mumkin.

Dasturlashda qator o'zgaruvchilar yaratiladi va ularga qiymatlar beriladi. Turlar yordamida dastur uchun qulay bo'lgan turli ko'rinishdagi qiymatlar aniqlanadi. Shunday qilib, turlarni dastur komponentlaridan biri deb aytish mumkin bo'ladi. Oddiy turlarga misol sifatida butun, uzun va suzuvchi turlarni keltirish mumkin. O'zgaruvchining turi ushbu o'zgaruvchi qanday qiymatlarni olishi va uning ustida qanday operatsiyalarni bajarish mumkinligini belgilab beradi.

Turlar dasturda qo'llash mumkin bo'lgan o'zgaruvchilar turini aniqlab beradi. Ushbu turdagi o'zgaruvchi qanday yo'l qo'yiladigan qiymatlarga ega bo'lishi mumkinligini tur belgilab beradi. Tur nafaqat yo'l qo'yiladigan qiymatlar sohasini, balki ushbu o'zgaruvchi ustida qanday operatsiyalarni bajarish mumkinligi, shuningdek, olinadigan natijalar qanday turda bo'lishligini ham belgilab beradi.

Turlar – hisoblarda bir butunlik sifatida amal qiladigan narsa. Masalan, butun sonni olaylik. Ikkita butun sonni qo'shar ekansiz, garchi bu sonlar kompyuter xotirasida bitlar ko'rinishida namoyon bo'lsa-da, siz bitlar ustidagi operatsiyalar haqida bosh qotirib o'tirmaysiz.

Joriy etish berkitilgani tufayli, obyekt ko'zda tutilmagan va destruktiv (tuzilmani buzadigan) foydalanishdan himoyalangan bo'ladi. Bu joriy qilish berkitilganligining afzalliklaridan biridir. Biroq joriy qilishning berkitilganligi obyektlardan foydalanuvchilar uchun ham muhimdir.

Joriy qilishning berkitilganligi dasturni moslashuvchan qiladi, chunki foydalanuvchilar obyektning joriy qilinishini hisobga olishga majbur emaslar. Shunday qilib, joriy qilishning berkitilganligi nafaqat obyektni himoyalaydi, balki kuchsiz bog'langan kodni yaratishga yordam berib, ushbu obyektdan foydalanuvchilar uchun muayyan noqulayliklarni chetlab o'tish imkonini beradi.

Kuchsiz bog'langan kod – bu boshqa komponentlarning joriy qilinishiga bog'liq bo'lmagan kod.

Kuchli bog'langan kod yoki bevosita aloqalarga ega kod – bu boshqa komponentlarning joriy qilinishi bilan uzviy bog'liq bo'lgan kod.

Inkapsulatsiyalash va joriy qilinishning berkitilganligi – mo'jiza emas. Interfeys o'zgartirilganda, eski interfeysga bog'liq bo'lgan eski kodni ham o'zgartirish kerak bo'ladi. Agar dasturni yozishda detallar interfeysda berkitilgan bo'lsa, buning natijasida kuchsiz bog'langan dastur yuzaga keladi.

Kuchli bog'langan dasturda inkapsulatsiyalashning afzalliklari yo'qoladi: mustaqil va takroran qo'llanadigan obyektlarning yaratilishi mumkin bo'lmaydi.

Joriy qilishning berkitilganligi o'z kamchiliklariga ham ega. Ba'zida interfeys yordamida olish mumkin bo'lganidan ko'proq axborot kerak bo'lib qoladi. Dasturlar olamida ma'lum aniqlik bilan, ya'ni ma'lum bir to'g'ri keladigan razryadlar miqdori bilan ishlaydigan qora qutilar kerak. Masalan, shunday vaziyat yuz berishi mumkinki, sizga 64 bitli

butun sonlar kerak bo'lib qoladi, chunki siz juda katta sonlar ustida amallar bajarayapsiz. Interfeysni belgilashda, uni taqdim etishgina emas, balki joriy qilishda qo'llanilgan turlarning o'ziga xos tomonlarini hujjatlashtirish ham g'oyat muhimdir. Biroq, ommaviy interfeysning har qanday boshqa qismi kabi, xulq-atvorni belgilagandan so'ng, uni o'zgartirib bo'lmaydi.

Joriy qilishni berkitib, mustaqil, boshqa komponentlar bilan kuchsiz bog'langan dasturni yozish mumkin. Kuchsiz bog'langan dastur mustahkamroq bo'ladi, bundan tashqari uni modifikatsiya qilish ham osonroq. Bular tufayli esa uni takroran qo'llash va takomillashtirish oson kechadi, chunki tizimning bitta qismidagi o'zgarishlar uning boshqa mustaqil qismlariga ta'sir qilmaydi.

Ma'suliyatning bo'linganligi. Joriy qilish berkitilganligining ma'suliyat tushunchasi bilan bog'liqligi tabiiydir. Kuchsiz bog'langan dasturni yaratish uchun, ma'suliyatni tegishli ravishda taqsimlash ham muhim. Ma'suliyat tegishli ravishda taqsimlanganda, har bir obyekt o'zi ma'sul bo'lgan bitta funksiyani bajaradi. Bu esa obyekt bir butunlikni tashkil etishini ham bildiradi. Boshqacha qilib aytganda, funksiyalar va o'zgaruvchilarning tasodifiy to'plamiga ehtiyoj bo'lmaydi. Inkapsulatsiyalanayotgan obyektlar o'rtasida yaqin konseptual aloqa bo'lmog'i kerak. Barcha funksiyalar umumiy vazifani bajarmog'i kerak.

Joriy qilish berkitilmas ekan, ma'suliyat obyektdan chetga chiqib ketishi mumkin. Biroq o'z vazifasini qanday hal qilishni aynan obyektning o'zi bilishi lozim, ya'ni aynan obyekt o'z vazifasini bajarish algoritmiga ega bo'lishi kerak. Agar joriy qilish ochiq qoldirilsa, foydalanuvchi undan to'g'ridan to'g'ri foydalanishi va shuning bilan ma'suliyatni bo'lishi mumkin.

Agar ikkita obyekt bir xil vazifani bajarsa, demak, ma'suliyat tegishlicha taqsimlanmagan bo'ladi. Dasturda ortiqcha mantiqiy sxemalar mavjud bo'lsa, uni qayta ishlash lozim bo'ladi.

Hayotda bo'lganidek, bilimlar va ma'suliyat ishni qanday qilib yaxshi bajarish mumkinligini bilgan kishiga vakolat qilinishi kerak. Bitta obyektga bitta (har holda kam miqdordagi) vazifa uchun ma'suliyatni yuklash lozim. Agar bitta obyektga ko'p miqdordagi vazifalar ustidan ma'suliyat yuklatib qo'yilgan bo'lsa, ularni bajarish murakkablashadi, obyektni kuzatib borish va takomillashtirish ham qiyinlashadi. Ma'suliyatni o'zgartirish ham xavfli, chunki bunda, agar obyekt bir necha xulq-atvor liniyalariga ega bo'lsa, ularni ham o'zgartirishga to'g'ri keladi. Natijada juda katta miqdordagi

axborot bir yerga jamlanib qoladi, uni esa teng taqsimlash lozim. Obyekt juda kattalashib ketgan hollarda, u amalda mustaqil dasturga aylanadi hamda protsedurali dasturlash afzalliklaridan foydalanish bilan birga uning barcha tuzoqlariga ham ilinib qolishi mumkin bo'ladi. Natijada siz inkapsulatsiyalash umuman qo'llanmagan dasturda yuzaga keladigan barcha muammolarga duch kelib qolasiz.

Obyekt bir-ikkitadan ortiq vazifa uchun mas'ul ekanini aniqlagach, ma'suliyatning bir qismini boshqa obyektga olib o'tish kerak.

Joriy etishning berkitilganligi – samarali inkapsulatsiyalash yo'lidagi qadamlardan biri, xolos. Ma'suliyatni tegishli ravishda taqsimlamay, siz protseduralar ro'yxatiga ega bo'lib qolasiz.

Samarali inkapsulatsiyalash=abstraksiya+joriy qilishning berkitilganligi+ma'suliyat.

Abstraksiyani olib tashlab, dasturdan takroran foydalanib bo'lmaydi. Joriy qilishning berkitilganligini olib tashlab, siz kuchli bog'langan dasturga ega bo'lasiz. Nihoyat, ma'suliyatni olib tashlash natijasida esa siz protsedurali, ma'lumotlar ishloviga mo'ljallangan, markazlashmagan kuchli bog'langan dasturga ega bo'lasiz.

Inkapsulatsiyalash: namunaviy xatolar. Abstraksiyani o'ta darajada qo'llash sinfni yozishda ma'lum muammolarni keltirib chiqarishi mumkin. Barcha foydalanuvchilarga hamda barcha vaziyatlarda birdek to'g'ri keladigan sinfni yozish mumkin emas.

Haddan tashqari abstraksiyalash ham xavfli bo'lishi mumkin. Hatto agar siz biror-bir elementning ishlab chiqilishida abstraksiyadan foydalangan bo'lsangiz, u shu bir elementda ham barcha vaziyatlarda ishlay olmasligi mumkin. Foydalanuvchining barcha ehtiyojlarini qondira oladigan sinfni yaratish juda qiyin. Abstraksiyaga o'ralashib qolish kerak emas, birinchi galda qo'yilgan masalani yechish kerak.

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

Masalan, bahaybat hisoblar yoki murakkab modelga o'xshash ancha murakkab masalalar ham uchraydi. Bu o'rinda gap ma'suliyatni taqsimlash nuqtayi nazaridan murakkablik haqida bormoqda. Obyektning ma'suliyat sohalari qancha ko'p bo'lsa, u shuncha murakkabroq bo'ladi va uni qo'llab-quvvatlash ham ancha murakkablik tug'diradi.

Va, nihoyat, dasturlashda abstraksiyalashdan foydalanishga o'rganish uchun vaqt kerak. Haqiqiy abstrakt dastur haqiqiy hayot talablariga asoslangan bo'lmog'i lozim. U dasturchi shunchaki takroran qo'llanadigan obyektni yaratishga jazm qilganligi natijasida yuzaga kelmaydi. Aytganlaridek, ixtiroga ehtiyoj tug'ilganidagina u tug'iladi. Xuddi shu tamoyil obyektlarni yaratishda ham amal qiladi. Birinchi martadayoq haqiqatan abstrakt, takroran qo'llanadigan obyektni yozish mumkin emas. Odatda, takroran qo'llaniladigan obyektlar ishda sinovdan o'tib bo'lgan hamda ko'plab o'zgarishlarga uchragan dasturni takomillashtirish jarayonida yaratiladi.

Ichki o'zgaruvchilarni hamma vaqt berkitish kerak: ular konstantalar bo'lgan holatlar bundan mustasno. Muhimi, ular nafaqat berkitilgan bo'lishi lozim, balki ularga faqat sinfning o'zi kirish huquqiga ega bo'lishi kerak. Ichki o'zgaruvchilarga kirishga ruxsat berilganda, joriy qilish ochiladi.

Ichki ma'lumotlari boshqa nom ostida tashqi foydalanish uchun taqdim etilgan interfeysni yaratishga ehtiyoj yo'q. Interfeys oliy darajadagi xulq-atvor yo'llariga ega bo'lishi lozim.

11.5. Vorislik

Vorislik ta'rifi. Vorislik bu mavjud sinflarga yangi maydonlar, xossalar va usullar qo'shish yordamida yangi sinflar hosil qilish imkoniyatini beradi. Yangi hosil qilingan avlod sinf asos, ya'ni ajdod sinf xossalari va usullariga vorislik qiladi.

Yangi berilganlar turi (sinf) oldindan mavjud bo'lgan sinfni kengaytirishdan hosil bo'ladi. Bunda yangi sinf oldingi sinfning merosxo'ri deb ataladi.

Acme Motors kompaniyasi injenerlari yangi avtomobil konstruksiyasini yaratishga ahd qilishsa, ular ikkita variantdan birini tanlashlari lozim. Birinchisi, avtomobilning konstruksiyasini boshidan boshlab yangidan ixtiro qilish, ikkinchisi esa mavjud Star modelini o'zgartirishdir. Star modeli qariyb ideal, faqatgina unga turbokompressor va olti tezlanishli uzatma qo'shish lozim. Bosh muhandis ikkinchi variantni tanladi. Ya'ni noldan boshlab qurishni emas, balki Star avtomobiliga ozgina o'zgartirish qilish orqali yaratishni tanladi. Uni yangi imkoniyatlar bilan rivojlantirmoqchi bo'ldi. Shuning uchun, yangi modelni Quasar deb nomlashni taklif qildi. Quasar Star modeliga yangi detallarni qo'shish orqali yaratilgan.

Vorislikdan foydalanish. Vorislik mavjud bo'lgan sinfning ta'rifi asosidayoq yangi sinfni yaratish imkonini beradi. Yangi sinf boshqasi asosida yaratilgach, uning ta'rifi avtomatik tarzda mavjud sinfning barcha xususiyatlari, xulq-atvori va joriy qilinishiga vorislik qiladi. Avval mavjud bo'lgan sinf interfeysining barcha usullari va xususiyatlari avtomatik tarzda voris interfeysida paydo bo'ladi. Vorislik voris sinfida biror-bir jihatdan to'g'ri kelmagan xulq-atvorni avvaldan ko'ra bilish imkonini beradi. Bunday foydali xususiyat dasturiy ta'minot talablarining o'zgarishiga moslashtirish imkonini beradi. Agar o'zgartirishlar kiritishga ehtiyoj tug'ilsa, bu holda eski sinf funksiyalariga vorislik qiluvchi yangi sinf yozib qo'ya qolinadi. Keyin o'zgartirilishi lozim bo'lgan funksiyalarga qaytadan ta'rif beriladi hamda yangi funksiyalar qo'shiladi. Bunday o'rniga o'rin qo'yishning mazmuni shundan iboratki, u dastlabki sinf ta'rifini o'zgartirmay turib, obyekt ishini o'zgartirish imkonini beradi. Axir bu holda qayta test sinovlaridan puxta o'tkazilgan asosiy sinflarga tegmasa ham bo'ladi.

Egalik («has») va bir turlilik («is a») munosabatlari. Odatda, sinflarni loyihalashda savol kelib chiqadi, sinflarning o'zaro munosabatini qanday qurish kerak bo'ladi. Ikkita oddiy sinflarga misol ko'ramiz – Square va Rectangle, ular kvadrat va to'g'ri to'rtburchaklardir. Shunisi tushunarliki, bu sinflar vorislik bog'lanishida bo'ladi, lekin ikkita sinfdan qaysi biri ajdod sinf bo'ladi. Yana ikkita sinfga misol – Car va Person, ya'ni mashina va inson. Bu sinflar bilan Person_of_Car, ya'ni mashina egasi sinfi qanday aloqada bo'lishi mumkin? Bu ikki sinf bilan vorislik bog'lanishida bo'lishi mumkinmi? Sinflarni loyihalash bilan bog'liq bu savollarga javob topish uchun shuni nazarda tutish kerakki, «mijoz-yetkazuvchi» bog'lanishi «ega» («has») bog'lanishini, vorislik bog'lanishi esa «bir xil» («is a») bog'lanishi tushunchalarini ifodalaydi. Square va Rectangle sinflari misoli tushunarli, har bir obyekt kvadrat to'g'ri to'rtburchakdir, shuning uchun bu sinflar o'rtasida vorislik bog'lanishi ifodalanadi va Rectangle sinfi ota-onalar sinfini ifodalaydi. Square sinfi uning o'g'lidir. Mashina egasi mashinaga ega va insondir. Shuning uchun Person_of_Car sinfi Car sinfning mijozi bo'lib hisoblanadi va Person sinfning vorisidir.

Vorislik tabaqalanishi qandaydir ma'no kasb etishi uchun ajdodlar ustidan qanday amallar bajarilgan bo'lsa, avlodlar ustidan ham shunday amallar bajarilish imkoniyati bo'lishi lozim. Merosxo'r sinfga funksiyalarni kengaytirish va yangilarini qo'shish uchun ruxsat beriladi. Ammo unga funksiyalarni chiqarib tashlashga ruxsat yo'q.

Vorislik yordamida qurilgan sinf usullar va xususiyatlarning uchta ko'rinishiga ega bo'lishi mumkin:

– o'rniga o'rin qo'yish (almashtirish): yangi sinf ajdodlarining usuli yoki xususiyatini shunchaki o'zlashtirib olmaydi, balki unga yangi ta'rif ham beradi;

– yangi: yangi sinf butunlay yangi usullar yoki xususiyatlarni qo'shadi;

– rekursiv: yangi sinf o'z ajdodlari usullari yoki xususiyatlarini to'g'ridan to'g'ri olib qo'ya qoladi.

Obyektga mo'ljallangan tillarning ko'pchiligi ta'rifni ma'lumot uzatilgan obyektdan qidiradilar. Agar u yerdan ta'rif topishning iloji bo'lmaca, biror ta'rif topilmaguncha, qidiruv tabaqalar bo'yicha yuqoriga ko'tarilaveradi. Ma'lumotni boshqarish aynan shunday amalga oshiriladi hamda aynan shu tufayli o'ringa o'rin qo'yish jarayoni ish ko'rsatadi.

Voris sinflar himoyalangan kirish darajasiga ega bo'lgan usullar va xususiyatlarga kirish huquqini olishlari mumkin. Bazaviy sinfda faqat avlodlar foydalanishi mumkinligi aniq bo'lgan usullargagina himoyalangan kirish darajasini bering. Boshqa hollarda xususiy yoki ommaviy kirish darajasidan foydalanish lozim. Bunday yondashuv barcha sinflarga, shu jumladan, tarmoq sinflarga ham kirish huquqi berilganidan ko'ra, mustahkamroq konstruksiyani yaratish imkonini beradi.

Vorislik turlari. Vorislik uch asosiy hollarda qo'llaniladi:

1) ko'p martalab foydalanishda;
2) ajralib turish uchun;
3) turlarni almashtirish uchun.

Vorislikning ayrim turlaridan foydalanish boshqalaridan ko'ra afzalroq hisoblanadi. Vorislik yangi sinfga eski sinfning amalda qo'llanilishidan ko'p martalab foydalanish imkonini beradi. Kodni qirqib tashlash yoki kiritish o'rniga, vorislik kodga avtomatik tarzda kirishni ta'minlaydi, ya'ni kodga kirishda, u yangi sinfning bir qismidek olib qaraladi. Ko'p martalab qo'llash uchun vorislikdan foydalanar ekansiz, siz meros qilib olingan realizatsiya (joriy qilinish) bilan bog'liq bo'lasiz. Vorislikning bu turini ehtiyotkorlik bilan qo'llash lozim. Farqlash uchun vorislik faqat avlod-sinf va ajdod-sinf o'rtasidagi farqlarni dasturlash imkonini beradi. Farqlarni dasturlash g'oyat qudratli vositadir. Kodlash hajmining kichikligi va kodning oson boshqarilishi loyiha ishlanmasini osonlashtiradi. Bu holda kod satrlarini

kamroq yozishga to'g'ri keladiki, bu qo'shiladigan xatolar miqdorini ham kamaytiradi.

Almashtirish imkoniyati – OMY da muhim tushunchalardan biri. Merosxo'r sinfga uning ajdodi bo'lmish sinfga yuboriladigan xabarlarni yuborish mumkin bo'lgani uchun, ularning har ikkalasiga bir xil munosabatda bo'lishi mumkin. Aynan shuning uchun merosxo'r sinfni yaratishda xulq-atvorni chiqarib tashlash mumkin emas. Almashtirish imkoniyatini qo'llab, dasturga har qanday tarmoq turlarni qo'shish mumkin. Agar dasturda ajdod qo'llangan bo'lsa, bu holda u yangi obyektlardan qanday fodalanishni biladi.

11.6. Polimorfizm

OYD tillari bir xil nomdagi funksiya turli obyekt tomonidan ishlatilganda turli amallarni bajarish imkoniyatini ta'minlaydi. Bu funksiya va sinfning polimorfligi deb nomlanadi. Poli – ko'p, morfe – shakl degan ma'noni anglatadi. Polimorfizm – bu shaklning ko'p xilligidir. Bu tushunchalar bilan keyinchalik batafsil tanishamiz.

Agar inkapsulatsiyalash va vorislikni OMY ning foydali vositalari sifatida olib qarash mumkin bo'lsa, polimorfizm – eng universal va radikal vositadir. Polimorfizm inkapsulatsiyalash va vorislik bilan chambarchas bog'liq, boz ustiga, polimorfizmsiz OMY samarali bo'lolmaydi. Polimorfizm – OMY paradigmasida markaziy tushunchadir. Polimorfizmni egallamay turib, OMY dan samarali foydalanish mumkin emas.

Polimorfizm shunday holatki, bunda qandaydir bitta narsa ko'p shakllarga ega bo'ladi. Dasturlash tilida «ko'p shakllar» deyilganda, bitta nom avtomatik mexanizm tomonidan tanlab olingan turli kodlarning nomidan ish ko'rishi tushuniladi. Shunday qilib, polimorfizm yordamida bitta nom turli xulq-atvorni bildirishi mumkin.

Vorislik polimorfizmning ayrim turlaridan foydalanish uchun zarurdir. Aynan o'rindoshlik imkoniyati mavjud bo'lgani uchun, polimorfizmdan foydalanish mumkin bo'ladi. Polimorfizm yordamida tizimga to'g'ri kelgan paytda qo'shimcha funksiyalarni qo'shish mumkin. Dasturni yozish paytida hatto taxmin qilinmagan funksionallik bilan yangi sinflarni qo'shish mumkin, buning ustiga bularning hammasini dastlabki dasturni o'zgartirmay turib ham amalga oshirish mumkin. Yangi talablarga osongina moslasha oladigan dasturiy vosita deganda, shularni keltirish mumkin.

Polimorfizmning uchta asosiy turi mavjud:

- qo'shilish polimorfizmi;

- parametrik polimorfizm;
- qo'shimcha yuklanish.

Qo'shilish polimorfizmini ba'zida sof polimorfizm deb ham ataydilar. Qo'shilish polimorfizmi shunisi bilan qiziqarliki, u tufayli tarmoq sinf nusxalari o'zini turlicha tutishi mumkin. Qo'shilish polimorfizmidan foydalanib, yangi tarmoq sinflarni kiritgan holda, tizimning xulq-atvorini o'zgartirish mumkin. Uning bosh afzalligi shundaki, dastlabki dasturni o'zgartirmay turib, yangi xulq-atvorni yaratish mumkin.

Aynan polimorfizm tufayli joriy qilishdan takroran ifodalanishni vorislik bilan aynanlashtirish kerak emas. Buning o'rniga vorislikdan avvalambor o'zaro almashinish munosabatlari yordamida polimorf xulq-atvorga erishish uchun foydalanish lozim. Agar o'zaro almashinish munosabatlari to'g'ri belgilansa, buning ortidan albatta takroran qo'llash chiqib keladi. Qo'shilish polimorfizmidan foydalanib, bazaviy sinfdan, har qanday avloddan, shuningdek, bazaviy sinf qo'llaydigan usullardan takroran foydalanish mumkin.

Parametrik polimorfizmdan foydalanib, turdosh usullar va turdosh (universal) turlar yaratish mumkin. Turdosh usullar va turlar dalillarning ko'plab turlari bilan ishlay oladigan dasturni yozish imkonini beradi. Agar qo'shilish polimorfizmidan foydalanish obyektni idrok etishga ta'sir ko'rsatsa, parametrik polimorfizmdan foydalanish qo'llanilayotgan usullarga ta'sir ko'rsatadi. Parametrik polimorfizm yordamida, parametr turini bajarilish vaqtigacha e'lon qilmay turib, turdosh usullar yaratish mumkin. Usullarning parametrik parametrlari bo'lganidek, turlarning o'zi ham parametrik bo'lishi mumkin. Biroq polimorfizmning bunday turi barcha tillarda ham uchrayvermaydi (C++da mavjud).

Qo'shimcha yuklanish yordamida bitta nom turlicha usullarni bildirishi mumkin. Bunda usullar faqat miqdorlari va parametr turlari bilan farqlanadi. Usul o'z dalillari (argumentlari)ga bog'liq bo'lmaganda, ortiqcha yuklanish foydalidir. Usul o'ziga xos parametrlar turlari bilan cheklanmaydi, balki har xil turdagi parametrlarga nisbatan ham qo'llanadi. Masalan, max usulini ko'rib chiqaylik. Maksimal – turdosh tushuncha bo'lib, u ikkita muayyan parametrlarni qabul qilib, ularning qaysi biri kattaroq ekanini ma'lum qiladi. Ta'rif butun sonlar yoki suzuvchi nuqtali sonlar qiyoslanishiga qarab o'zgarmaydi.

Polimorfizmdan samarali foydalanish sari qo'yilgan birinchi qadam bu inkapsulatsiyalash va vorislikdan samarali foydalanishdir. Inkapsulatsiyalashsiz dastur osongina sinflarning joriy qilinishiga bog'liq bo'lib qolishi

mumkin. Agar dastur sinflarning joriy qilinish aspektrlaridan biriga bog'liq bo'lib qolsa, tarmoq sinfda bu joriyni to'g'rilash mumkin bo'lmaydi.

Vorislik – qo'shilish polimorfizmining muhim tarkibiy qismi. Hamma vaqt bazaviy sinfga imkon darajada yaqinlashtirilgan darajada dasturlashga uringan holda, o'rinbosarlik munosabatlarini o'rnatishga harakat qilish kerak. Bunday usul dasturda ishlov berilayotgan obyektlar turlari miqdorini oshiradi.

Puxta o'ylab ishlab chiqilgan tabaqalanish o'rinbosarlik munosabatlarini o'rnatishga yordam beradi. Umumiy qismlarni abstrakt sinflarga olib chiqish kerak hamda obyektlarni shunday dasturlash kerakki, bunda obyektlarning ixtisoslashtirilgan nusxalari emas, balki ularning o'zlari dasturlashtirilsin. Bu keyinchalik har qanday voris sinfni dasturda qo'llash imkonini beradi.

Agar til vositalari bilan interfeys va joriy qilinishni to'liq ajratish mumkin bo'lsa, u holda, odatda, mana shu vositalardan foydalanish kerak, vorislikdan emas. Interfes va joriy qilinishni aniq ajratib, o'rinbosarlik imkoniyatlarini oshirish va shuning bilan polimorfizmdan foydalanishning yangi imkoniyatlarini ochib berish mumkin.

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

Akseleratorning bosilishida Star modeliga nisbatan yangi yaratilgan Quasar modelida boshqacharoq amallar bajarilishi mumkin. Quasar modelida dvigatelga yoqilg'ini sepuvchi injektor sistemasi va Star modelidagi korbyurator o'rniga turbokompressor o'rnatilgan bo'lishi mumkin. Lekin foydalanuvchi bu farqlarni bilishi shart emas. U rulga o'tirgach oddiygina akselatorni bosadi va avtomobilning mos reaksiyasini kutadi.

12-bob. TIL ASOSLARI

12.1. O'zgaruvchilar va konstantalar

O'zgaruvchilar. O'zgaruvchi nomi ostiga chizish belgisi yoki lotin harfidan boshlanuvchi lotin harflari, arab raqamlari va ostiga chizish belgilari ketma-ketligi ya'ni identifikatordir.

O'zgaruvchilarning quyidagi turlari mavjud: **char** (simvol), **bool** (mantiqiy), **short** (qisqa butun), **int** (butun), **long** (uzun butun), **float** (haqiqiy), **double** (ikkilangan haqiqiy).

Butun sonlar ta'riflanganda qurilgan turlar oldiga **unsigned** (ishorasiz) ta'rifi qo'shilishi mumkin. Ishorali, ya'ni **signed** turidagi sonlarning eng katta razryadi son ishorasini ko'rsatish uchun ishlatilsa, **unsigned** (ishorasiz) turdagi sonlarda bu razryadli sonni tasvirlash uchun ishlatiladi

O'zgaruvchilar ta'rifining sodda shakli:

<tur> <o'zgaruvchilar_nomlari_ro'yxati >;

o'zgaruvchilarni ta'riflashda boshlang'ich qiymatlarini ko'rsatish mumkin.

<tur> <o'zgaruvchilar_nomlari_ro'yxati >=<initsializator>;

Bu usul initsializatsiya deyiladi.

Misollar:

float pi=3.14, cc=1.3456;

unsigned int year=1999;

Konstantalar. Konstanta – bu o'zgartirish mumkin bo'lmagan qiymatdir. Konstantalarning butun, simvolli, haqiqiy va mantiqiy turlari mavjud.

Butun sonlar o'nlik, sakkizlik yoki o'n oltilik sanoq sistemalarida berilishi mumkin:

– o'nlik konstantalar o'nlik raqamlar ketma-ketligidan iborat bo'lib, birinchi raqami 0 bo'lishi kerak emas (masalan: 8, 0, 192345);

– sakkizlik konstantalar 0 bilan boshlanuvchi sakkizlik raqamlardan iborat ketma-ketlikdir (masalan: 016 – o'nlik qiymati 14, 01);

– o'n oltilik konstantalar – 0x yoki 0X bilan boshlanuvchi o'n oltilik raqamlar ketma-ketligidir (masalan: 0xA, 0X00F).

Simvolli konstanta – apostrifga olingan bitta (masalan: 'g', 'r', '6') yoki bir nechta (masalan: – '\n', '\0xF5') simvol. Slesh '\' simvolidan boshlangan simvollar eskeyp yoki boshqaruvchi simvollar deyiladi.

Haqiqiy konstanta ikki ko'rinishda bo'lishi mumkin: fiksirlangan nuqtali (masalan: 5.7, .0001, 41.) va suzuvchi nuqtali (masalan: 0.5e5, .11e-5).

Mantiqiy konstantalar **true**(rost) va **false**(yolg'on) qiymatlardan iborat. Ichki ko'rinishi **false** – 0, ixtiyoriy boshqa qiymat **true** deb qaraladi.

Nomlangan konstantalar. Nomlangan konstantalar quyidagi shaklda kiritiladi:

const tur konstanta_nomi=konstanta_qiymati.

Ko'zda tutilgani bo'yicha **int** turidan foydalaniladi.

Misol uchun:

const double EULER=2.718282;

const long M=99999999;

const R=765;

Yangi tur. Ta'riflovchi typedef yangi turlarni kiritishga imkon beradi. Misol uchun yangi **COD** turini kiritish:

Typedef unsigned char COD;

COD simbol;

Sanovchi tur. Agar har xil qiymatlarga ega bir nechta nomlangan konstanta kiritish kerak bo'lsa, sanovchi turdan foydalanish mumkin:

enum <tur_nomi > {<konstantalar ro'yxati >};

Konstantalar butun bo'lishi kerak va initsializatsiya qilinishi mumkin, agar initsializator mavjud bo'lmasa, birinchi konstanta nol qiymat oladi, qolganlari esa oldingisidan birga ortiq bo'ladi.

Misol uchun:

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

Agar son qiymatlari ko'rsatilmagan bo'lsa, eng chapki so'zga 0 qiymati berilib, qolganlariga tartib bo'yicha o'suvchi sonlar mos qo'yiladi:

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 qiymatlarni qabul qiladi:

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

Nomlangan sanovchi tur kiritib, shu turdagi o'zgaruvchilar, ko'rsatkichlar va ilovalardan foydalanish mumkin. Masalan:

enum color (black, green, yellow, blue, red, white);

color col=red;

color* cp=&col;

if (*cp = = green) cout<<"yashil";

12.2. Amallar

Arifmetik amallar. Arifmetik amallar binar va unar amallarga ajratiladi. Binar amallarga+qo'shish, – ayirish, * ko'paytirish, / bo'lish va % modul olish amallari kiradi.

Masalan, 20/3=6; (–20)/3=–6; 5%2=1;

Unar amallarga unar minus – va unar +; inkrement ++ va dekrement – amallari kiradi. Inkrement va dekrement amallari prefiks, ya'ni ++i, postfiks, ya'ni i++ ko'rinishda ishlatilishi mumkin. Masalan, agar i=2, k=2 bo'lsa, u holda 3+(++i)=6, 3+k++=5 ga teng bo'ladi. Ikkala holda ham i=3, k=3 ga teng bo'ladi.

Nisbat amallari. Nisbat amallari qiymati mantiqiy bo'lib, katta >; kichik <; katta yoki teng >=; kichik yoki teng <=; teng = =; teng emas != amallaridan iborat.

Mantiqiy amallar. Mantiqiy amallar || (dizyunksiya); && (konyunksiya); !(inkor) amallaridan iborat.

Razryadli amallar. Razryadli amallar | (razryadli dizyunksiya); & (razryadli konyunksiya); ^(razryadli XOR); !(razryadli inkor); chapga surish <<; o'ngga surish >> amallaridan iborat.

Masalan, 5 kodi 101 ga teng va 6 kodi 110 ga teng:

6&5=4=100; 6|5=7=111; 6^5=3=011.

5<<2=20 yoki 101<<2=10100; 5>>2=1 yoki 101>>2=001=1.

Qiymat berish amali. Oddiy qiymat berish amali binar amal bo'lib, chap operandi, odatda, o'zgaruvchi, o'ng operandi, odatda, ifodaga teng bo'ladi:

O'zgaruvchi_nomi=ifoda;

Masalan: z=4.7+3.34; c=y=f=4.2+2.8.

Murakkab qiymat berish amali unar amal bo'lib, quyidagi ko'rinishga ega:

O'zgaruvchi_nomi **amal**=ifoda;

Masalan: x+=4 ifoda x=x+4 ifodaga ekvivalent;

x>>=4 ifoda x=x>>4 ifodaga ekvivalent.

Shartli amal. Shartli amal ternar amal deyiladi va uchta operanddan iborat bo'ladi: <1-ifoda>?<2-ifoda>:<3-ifoda>. Masalan: **y=a<b?a:b.**

12.3. Dastur strukturasi

C++ tilidagi dastur quyidagi strukturaga ega:

#<preprotsessor direktivasi>

<funksiyalar>

void main () <operatorlar>

preprotsessor direktivasi – dasturdagi almashtirishlarni uning kompilatsiyasigacha boshqaradi.

1) **#define** – matndagi almashtirishlar qonun qoidasini aniqlaydi.

Misol:

#define ZERO 0.0

Bu dastur kodidagi har bir zero so'zi 0.0 bilan almashtirilishini bildiradi.

2) **#include<sarlavha fayl nomi>** – standart kutubxonalar bilan birga yetkaziladigan sarlavha faylidagi funksiyalarni ishlatishga mo'ljallangan.

Masalan, Hello so'zini chop etish:

```
#include<iostream.h>
void main() {
cout<<"Hello";
}
```

Hozirgi davrda bunday yozuv eskirgan. Zamonaviy variant:

```
#include<iostream>
using namespace std;
int main() {
cout<<«Hello»;
return 0;
}
```

12.4. Operatorlar

Ifoda operatori. Har bir ifoda agar u nuqta vergul belgisi bilan tugasa, operator hisoblanadi. Masalan: **int i=7+k;**

Izoh (kommentariya) operatori. Izoh operatori bajarilmaydigan operator bo'lib, ko'p satrli (masalan **/*bu izoh */**) yoki bir satrli (masalan, **//bu izoh)** bo'lishi mumkin. Izohlar /* va */ belgilar bilan ajratilgan bo'lsa, ichki joylashgan bo'la olmaydi. Agar kod fragmentini /* va * / belgilar bilan ishlamaydigan qilib bo'lmaydi, chunki uning o'zi */ va */ simvollarni o'z ichiga olishi mumkin.

Blok. Figurali qavsga olingan operatorlar va ta'riflar ketma-ketligi blok deyiladi. Masalan: **{int i, s; i=0; s=0;};**

Kiritish chiqarish operatorlari. Ma'lumotni kiritish uchun *cin*, chiqarish uchun *cout* operatoridan foydalaniladi. Kursorni keyingi qatorga o'rnatish uchun *endl* (qator oxiri) simvoli yoki yangi qator simvoli (*\n*)dan foydalanish mumkin.

Masalan: **cin>>i>>k;** yoki **cin>>i; cin>>k;**

cout<<g<<h; yoki **cout<<g; cout<<h;**

Tanlash operatorlari.

Shartli operator. Shartli tanlash operatorida avval shart tekshiriladi. Agar shart rost bo'lsa, birinchi, aks holda ikkinchi operator (agar u bo'lsa) bajariladi.

if (ifoda) 1-operator else 2-operator yoki **if (ifoda) 1-operator**

Misol. Simvol ikkilik ekanligini tekshiruvchi dastur:

```
#include<iostream>
using namespace std;
int main() {
char ch='1';
if (ch=='1' || ch=='0') cout<< "Simvol binary";
else cout<<"Simvol no binary";
return 0;
}
```

Kalit bo'yicha tanlash operatori. Kalit bo'yicha tanlash operatori quyidagi shaklga ega:

```
switch(<ifoda>) {
case <1-qiymat>:<1-operator>
...
default: <operator>
...}
```

Kalit bo'yicha tanlash operatorida berilgan ifoda qiymati biror case qiymatiga mos kelsa, shundan keyingi hamma operatorlar bajariladi, aks holda **default** (agar u bo'lsa) so'zidan keyingi operator bajariladi.

Misol. Simvol ikkilik ekanligini tekshiruvchi dastur:

```
#include<iostream>
using namespace std;
int main() {
char ch='5';
switch (ch) {
case '0':
case '1': cout<< "Simvol binary"; break;
default: cout<<"Simvol no binary";
}
return 0;
}
```

Sikl operatorlari. *Oldingi shartli sikl* quyidagi ko'rinishga ega:

```
while (<shart>) <sikl tanasi>;
```

Oldingi shartli siklda oldin shart tekshiriladi keyin to shart yolg'on bo'lguncha sikl tanasi bajariladi.

Misol. Sikl yordamida o'ngacha sonlar yig'indisini hisoblash:

```
#include<iostream>
using namespace std;
int main() {
int i=1, s=0;
while(i<=10) s+=i++;
cout<<s;
return 0;
}
```

Keyingi shartli sikl quyidagi ko'rinishga ega:

```
do
<sikl tanasi>;
while (<shart>);
```

Oldin sikl tanasi bajarilib, keyin shart tekshiriladi. Sikl to shart yolg'on bo'lmaguncha davom etadi.

Misol. Sikl yordamida o'ngacha sonlar yig'indisini hisoblash:


```
#include<iostream>
using namespace std;
int main() {
int i=1, s=0;
do s+=i++; while(i<=10);
cout<<s;
return 0;
}
```

Parametrli sikl quyidagi ko'rinishga ega:

```
for(1-ifoda; shart; 2-ifoda)
sikl tanasi;
```

Avval 1-ifoda bajariladi va to shart yolg'on bo'lmaguncha sikl tanasi va 2-ifoda bajariladi. Ixtiyoriy ifoda bo'sh bo'lishi mumkin, lekin ularni ajratuvchi qavs «;» bo'lishi shart.

Misol. Sikl yordamida o'ngacha sonlar yig'indisini hisoblash:

```
#include<iostream>
using namespace std;
int main() {
int s=0;
for(int i=1; i<=10; i++) s+=i;
cout<<s;
return 0;
}
```

Siklda bir necha parametrning qo'llanishi:

```
#include <iostream>
using namespace std;
int main()
{
for (int i=0, j=0; i<3; i++, j++)
cout<< "i:" <<i<< "j:" <<j<< endl;
return 0;
}
```

Sikl tanasi bo'sh bo'lgan hol:

```
#include <iostream>
```

```
using namespace std;
int main()
{
for (int i=0; i<5; cout<< "i" <<i++<<endl);
return 0;
}
```

O'tish operatorlari. O'tish operatori boshqarishni shartsiz uzatishni amalga oshiradi.

Agar siklni davom ettirish shartini sikl o'rtasida tekshirish zarur bo'lsa, **break** operatoridan foydalanish qulay.

Masalan, cheksiz sikldan **break** operatori yordamida chiqish:

```
#include<iostream>
using namespace std;
int main( ) {
int i=1, s=0, n=3;
while(1) {if(i>n) break; s+=i++;};
cout<<s;
return 0;
}
```

Shu dastur for operatori yordamida:

```
#include<iostream>
using namespace std;
int main() {
int i=1, s=0, n=3;
for(;;) {if(i>n) break; s+=i++;};
cout<<s;
return 0;
}
```

Siklda keyingi iteratsiyasiga o'tish uchun **continue** operatoridan foydalaniladi.

Masalan, uchga va beshga karrali bo'lmagan sonlar yig'indisini hisoblash:

```
#include<iostream>
using namespace std;
```



```
int main() {
int s=0, n=7;
for(int i=1; i<=n; i++){
if ((i%3==0)||(i%5==0)) continue;
s+=i;
}
cout<<s;
return 0;
}
```

12.5. Foydalanuvchi funksiyalari

Funksiya ta'rifi. Funksiyani C++ tilida quyidagi ikki ko'rinishda qarash mumkin:

- hosila turlardan biri;
- dastur bajariluvchi minimal moduli.

Funksiya ta'rifining umumiy ko'rinishi quyidagicha:

<tur> <funksiya nomi>(<formal_parametrlar_ta'rifi>)

Formal parametrlarga ta'rif berilganda ularga boshlang'ich qiymatlari ham ko'rsatilishi mumkin.

Funksiya qaytaruvchi ifoda qiymati funksiya tanasida **return** <ifoda>; operatori orqali ko'rsatiladi.

Funksiyaga murojaat qilish quyidagicha amalga oshiriladi:

<Funksiya nomi> (<haqiqiy parametrlar ro'yxati>)

Masalan:

```
#include<iostream>
using namespace std;
float min(float a, float b)
{
if (a<b) return a; else return b;
}
int main()
{
float y=6.0, z;
z=min(3.3, y);
cout<<z;
return 0;
}
```

```
using namespace std;
int main()
{
for (int i=0; i<5; cout<< "i" <<i++<<endl);
return 0;
}
```

O'tish operatorlari. O'tish operatori boshqarishni shartsiz uzatishni amalga oshiradi.

Agar siklni davom ettirish shartini sikl o'rtasida tekshirish zarur bo'lsa, **break** operatoridan foydalanish qulay.

Masalan, cheksiz sikldan **break** operatori yordamida chiqish:

```
#include<iostream>
using namespace std;
int main( ) {
int i=1, s=0, n=3;
while(1) {if(i>n) break; s+=i++;};
cout<<s;
return 0;
}
```

Shu dastur for operatori yordamida:

```
#include<iostream>
using namespace std;
int main() {
int i=1, s=0, n=3;
for(;;) {if(i>n) break; s+=i++;};
cout<<s;
return 0;
}
```

Siklda keyingi iteratsiyasiga o'tish uchun **continue** operatoridan foydalaniladi.

Masalan, uchga va beshga karrali bo'lmagan sonlar yig'indisini hisoblash:

```
#include<iostream>
using namespace std;
```



```
int main() {
int s=0, n=7;
for(int i=1; i<=n; i++){
if ((i%3==0)||(i%5==0)) continue;
s+=i;
}
cout<<s;
return 0;
}
```

12.5. Foydalanuvchi funksiyalari

Funksiya ta'rifi. Funksiyani C++ tilida quyidagi ikki ko'rinishda qarash mumkin:

- hosila turlardan biri;
- dastur bajariluvchi minimal moduli.

Funksiya ta'rifining umumiy ko'rinishi quyidagicha:

<tur> <funksiya nomi>(<formal_parametrlar_ta'rifi>)

Formal parametrlarga ta'rif berilganda ularga boshlang'ich qiymatlari ham ko'rsatilishi mumkin.

Funksiya qaytaruvchi ifoda qiymati funksiya tanasida **return** <ifoda>; operatori orqali ko'rsatiladi.

Funksiyaga murojaat qilish quyidagicha amalga oshiriladi:

<Funksiya nomi> (<haqiqiy parametrlar ro'yxati>)

Masalan:

```
#include<iostream>
using namespace std;
float min(float a, float b)
{
if (a<b) return a; else return b;
}
int main()
{
float y=6.0, z;
z=min(3.3, y);
cout<<z;
return 0;
}
```

Funksiya ta'rifida formal parametrlar initsializatsiya qilinishi, ya'ni boshlang'ich qiymatlar ko'rsatilishi mumkin. Bu parametrlar initsializatsiya qilinmagan parametrlardan keyin kelishi shart.

Misol uchun:

```
#include<iostream>
using namespace std;
float min(float a, float b=0.0)
{
if (a<b) return a; else return b;
}
int main()
{
float y=6.0, z;
z=min(y);
cout<<z;
return 0;
}
```

Agar funksiya hech qanday qiymat qaytarmasa, uning turi **void** deb ko'rsatiladi.

Misol uchun:

```
#include<iostream>
using namespace std;
void Print(int n)
{
for(int i=0; i<n; i++)
cout<<"Salom"<<endl;
}
int main()
{
Print(3);
return 0;
}
```

Bu dastur bajarilishi ekranga uch marta Salom! yozilishiga olib keladi.

Prototip. Agar dasturda funksiya ta'rifi murojaatdan keyin berilsa, yoki funksiya boshqa faylda joylashgan bo'lsa, murojaatdan oldin shu

funksiyaning prototipi joylashgan bo'lishi kerak. Prototip funksiya nomi va formal parametrlar turlaridan iborat bo'ladi. Formal parametrlar nomlarini berish shart emas.

Misol uchun:

```
#include<iostream>
using namespace std;
int main()
{
float min(float, float);
float y=6.0, z;
z=min(3.3, y);
cout<<z;
return 0;
};
float min(float a, float b)
{
if (a<b) return a;
return b;
}
```

Proseduralar. Funksiyaga parametrlar qiymat bo'yicha uzatiladi. Funksiyaga parametrlar qiymatlari uzatilishi haqiqiy parametrlar qiymatlarini funksiya tanasida o'zgartirish imkonini bermaydi. Bu muammoni hal qilish uchun ko'rsatkichlardan foydalanish mumkin.

Masalan:

```
#include<iostream>
using namespace std;
void change (int &a, int &b)
{
int r;
r=a; a=b; b=r;
}

int main()
{
int a=1, b=2;
change(a, b);
```

```
cout<<"a="<<a<<" b="<<b;
return 0;
};
```

Inlayn funksiyalar. Dasturda funksiya ta'riflanganda kompilator funksiya kodini bir marta mashina kodiga o'tkazadi va dasturga ma'lumotlarni stekka joylovchi instruksiyalar qo'shadi. Argumentlarni stekka qo'shish, funksiyaga o'tish va qaytish mashina vaqtini oladi.

C++ kompilatori inline, so'zini uchratsa, bajariluvchi faylga har bir funksiyaga murojaat o'rniga funksiya operatorlarini qo'yadi. Shunday qilib, dastur effektivligini oshirish mumkin.

Masalan:

```
#include<iostream>
#include<math>
using namespace std;
inline float Line(float x1, float y1, float x2=0, float y2=0)
{
return sqrt(pow(x1–x2, 2)+pow(y1–y2, 2));
}

int main()
{
float a=1.0, b=2.0;
float z=Line(a, b);
cout<<"z="<<z;
return 0;
};
```

Bu dasturda inlayn funksiya ikki nuqta orasidagi masofani qaytaradi.

Funksiyalarni qo'shimcha yuklash. Funksiyalarni qo'shimcha yuklashdan maqsad bir xil nomli funksiyaga har xil turli o'zgaruvchilar bilan murojaat qilib, qiymat olishdir. Kompilator haqiqiy parametrlar ro'yxati va funksiya chaqirig'i asosida qaysi funksiyani chaqirish kerakligini o'zi aniqlaydi.

Misol uchun har xil o'zgaruvchilarni ko'paytirish uchun quyidagi funksiyalar kiritilgan bo'lsin:

```
#include<iostream>
```



```
#include<math>
using namespace std;
float min(float a, float b) {
if (a<b) return a; else return b;
}
int min(int a, int b) {
if (a<b) return a; else return b;
}
int main()
{
int a=1;
float x=4.5;
int m=min(6, a);
float c=min(5.0, x);
cout<<"m="<<m<<" c="<<c;
return 0;
};
```

Funksiyalarni qo'shimcha yuklash qoidalari

1. Funksiyalar bitta ko'rinish sohasiga tegishli bo'lishi lozim.
2. Funksiyalar parametrlari ko'zda tutilgan qiymatlarga ega bo'lishi mumkin, lekin har xil funksiyalardagi bir xil parametrlar bir xil qiymatga ega bo'lishi kerak.
3. Agar funksiyalar parametrlari ta'rifi faqat const modifikatori yoki ilova mavjudligi bilan farq qilsa, bu funksiyalarni qo'shimcha yuklash mumkin emas.

Masalan, kompilator int&f1(int&, const int&){...} va
int f1(int, int){...} – funksiyalarni ajrata olmaydi.

Rekursiv funksiyalar. Rekursiv funksiya deb o'ziga o'zi murojaat qiluvchi funksiyaga aytiladi.

Misol uchun faktorialni hisoblash funksiyasini keltiramiz:

```
#include<iostream>
using namespace std;
int factorial(int n)
{
if (n != 1) return n * factorial(n-1);
```

```
else return 1;
}
int main()
{
cout<<factorial(3);
return 0;
}
```

12.6. Massivlar va satrlar

Massivlarni ta'riflash. Massiv indeksli o'zgaruvchidir.

Massivning sodda ta'rifi:

<tur> <o'zgaruvchi_nomi>[<konstanta_ifoda>]=<initsializator>;

Massivning indekslar qiymati har doim 0 dan boshlanadi.

Ko'p o'lchovli massiv initsializatsiya qilinganda massivning birinchi indeksi chegarasi ko'rsatilishi shart emas, lekin qolgan indekslar chegaralari ko'rsatilishi shart.

Misol uchun:

```
int a[6]; float b[8], c[100];
double d[]={1, 2, 3, 4, 5};
int A [20][10];
int A [30][20][10];
int A [3][3]={0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
int A[ ][3]={{0, 1, 100}, {200, 210, 300}, {1000, 2000, 2100}};
```

Satrlar. Satrli konstanta ikkilik qavslarga olingan simvollar ketmaketligidir. Satrli konstanta oxiriga avtomatik ravishda satr ko'chirish '\n' simvoli qo'shiladi.

Satr qiymati simvolli konstanta bo'lgan simvolli massiv sifatida ta'riflanadi.

Misol uchun:

```
char capital[]="TASHKENT";
char capital[]={'T', 'A', 'S', 'H', 'K', 'E', 'N', 'T', '\n'};
char A[ ][9]={"Tashkent", "Samarqand", "Xiva"};
```

Massivlar va satrlar funksiya parametrlari sifatida. Massivlar ilova bo'yicha uzatiladi, ya'ni ularning qiymati funksiyada o'zgarishi mumkin.

Misol:

```
//massiv elementlari summasini hisoblash
int sum (int n, int a[])
{int i, int s=0;
for(i=0; i<n; i++) s+=a[i];
return s;
}
```

Satrlar parametrlar sifatida char[] turidagi bir o'lchovli massivlar sifatida uzatilishi mumkin. Bu holda satr uzunligini aniq ko'rsatish shart emas. Misol:

```
//simvollar sonini hisoblash
int strlen (char a[])
{int i=0; while(a[i++]);
return i;
}
```

Funksiyalarda massivlar argument sifatida ishlatilganda ularning birinchi indeksi chegarasini ko'rsatish shart emas, qolganlarining chegarasini ko'rsatish shart.

Misol:

```
#include<iostream>
using namespace std;
int count_family (int n, char a[][100], char c[]) {
int s=0;
for(i=0; i<n; i++) if (strcmp(a[i], c)) C++; return s;}
int main() {
shar c[]="TASHKENT";
char a[ ][9]={"TASHKENT", "SAMARQAND", "XIVA"};
cout<< count_family (3, a, c);
return 0;
}
```

Satrli funksiyalar. Satrli funksiyalardan foydalanish uchun dasturga <string.h> sarlavhali faylni ulash lozim.

Satrda simvollar sonini hisoblash uchun strlen funksiyasidan foydalaniladi. strlen satrdagi simvollar sonini qaytaradi. Satr oxirini bildiruvchi noll simvol hisobga kirmaydi.

Satrdan nusxa olish uchun strcpy funksiyasidan foydalaniladi.

Funksiya strcpy satr simvollarini dest satrga nusxa oladi va dest satr qaytaradi.

Satrlarni ulash uchun strcat funksiyasidan foydalaniladi.

Birinchi dest satr oxiriga src satr simvollarini ulaydi.

Natija uzunligi strlen(dest)+strlen(src).

Satrlarni solishtirish uchun strcmp funksiyasidan foydalaniladi.

funksiya s1 va s2 satrlarni leksikografik solishtiradi.

Qaytaradigan qiymati:

natija<0, agar s1 <s2

natija = = 0, agar s1 = = s2

natija > 0, agar s1 > s2

12.7. Turlar bilan ishlash

Turlarni keltirish. Turlarni keltirish (type casting) ma'lum turdagi o'zgaruvchi boshqa turdagi qiymat qabul qilganda foydalaniladi. Ba'zi turlar uchun keltirish avtomatik ravishda bajariladi. Avtomatik turlarni keltirish o'zgaruvchi turi hajmi qiymatni saqlashga yetarli bo'lganda bajariladi. Bu jarayon kengaytirish *(widening)* yoki yuksaltirish *(promotion)* deb ataladi. Chunki kichik razryadli tur katta razryadli turga kengaytiriladi. Bu holda turlarni avtomatik keltirish xavfsiz deb ataladi. Masalan, int turi char turidagi qiymatni saqlashga yetarli, shuning uchun turlarni keltirish talab qilinmaydi. Teskari jarayon toraytirish *(narrowing)* deb ataladi, chunki qiymatni o'zgartirish talab etiladi. Bu holda turlarni avtomatik keltirish xavfli deb ataladi. Masalan, haqiqiy turni butun turga keltirilganda kasr qism tashlab yuboriladi.

Amallarda turlarni avtomatik keltirish. Binar arifmetik amallar bajarilganda turlarni keltirish quyidagi qoidalar asosida amalga oshiriladi:

- doim short va char turlari int turiga keltiriladi;
- agar operandlardan biri long turiga tegishli bo'lsa, ikkinchi operand ham long turiga keltiriladi va natija ham long turiga tegishli bo'ladi;
- agar operandlardan biri float turiga tegishli bo'lsa, ikkinchi operand ham float turiga keltiriladi va natija ham float turiga tegishli bo'ladi;
- agar operandlardan biri double turiga tegishli bo'lsa, ikkinchi operand ham double turiga keltiriladi va natija ham double turiga tegishli bo'ladi;
- agar operandlardan biri long double turiga tegishli bo'lsa, ikkinchi operand ham long double turiga keltiriladi va natija ham long double turiga tegishli bo'ladi.

Ifodalarda turlarni avtomatik keltirish. Agar ifodada short va int turidagi o'zgaruvchilar ishlatilsa, butun ifoda turi int ga ko'tariladi. Agar ifodada biror o'zgaruvchi turi long bo'lsa, butun ifoda turi long turga ko'tariladi. Ko'zda tutilgan bo'yicha hamma butun konstantalar int turiga ega deb qaraladi. Hamma butun konstantalar oxirida L yoki l simvoli turgan bo'lsa, long turiga ega.

Agar ifoda float turidagi operandga ega bo'lsa, butun ifoda float turiga ko'tariladi. Agar biror operand double turiga ega bo'lsa, butun ifoda double turiga ko'tariladi.

Turlar bilan ishlovchi amallar. Turlarni o'zgartirish amali quyidagi ko'rinishga ega:

(tur_nomi) operand;

Bu amal operandlar qiymatini ko'rsatilgan turga keltirish uchun ishlatiladi. *Operand sifatida konstanta, o'zgaruvchi yoki qavslarga olingan* ifoda kelishi mumkin. Misol uchun (long)6 amali konstanta qiymatini o'zgartirmagan holda operativ xotirada egallagan baytlar sonini oshiradi. Bu misolda konstanta turi o'zgarmagan bo'lsa, (double)6 yoki (float)6 amali konstanta ichki ko'rinishini ham o'zgartiradi. Katta butun sonlar haqiqiy turga keltirilganda, sonning aniqligi yo'qolishi mumkin.

Masalan:

int x=1.7+1.8;

int y=(int)1.7+(int)1.8;

Bu amallar bajarilishi natijasida x o'zgaruvchi qiymati 3 ga, y o'zgaruvchi qiymati ikkiga teng bo'ladi.

Turlarni o'zgartirish amalining zamonaviy shakli quyidagi ko'rinishga ega:

Funksional: tur_nomi (operand); masalan: **z=double(1);**

Lekin bu variantni faqat bir so'zli turlarga qo'llash mumkin. Masalan, long double turiga qo'llab bo'lmaydi.

Xotiradagi hajmni hisoblash **sizeof** amalining ikki ko'rinishi mavjud:

sizeof ifoda masalan: sizeof (3.14)=8

sizeof (tur) masalan: sizeof(char)=1

Shuni ta'kidlab o'tish lozimki, sizeof funksiyasi preprotsessor qayta ishlash jarayonida bajariladi, shuning uchun dastur bajarilish jarayonida vaqt talab etmaydi.

Misol uchun:

sizeof 3.14=8

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

Lokal va global o'zgaruvchilar. C++ tilida o'zgaruvchi ta'rifi albatta blok boshida joylashishi shart emas.

O'zgaruvchi mavjudlik sohasi deb, shu o'zgaruvchiga ajratilgan xotira mavjud bo'lgan dastur qismiga aytiladi. O'zgaruvchi ko'rinish sohasi deb, o'zgaruvchi qiymatini olish mumkin bo'lgan dastur qismiga aytiladi. Biror blokda ta'riflangan o'zgaruvchi lokal o'zgaruvchi deyiladi. Har qanday blokdan tashqarida ta'riflangan o'zgaruvchi global o'zgaruvchi deyiladi.

Lokal o'zgaruvchi mavjudlik va ko'rinish sohasi ta'rifdan, to shu ta'rif joylashgan blok oxirigacha bajariladi.

Tashqi blokdagi o'zgaruvchi nomi shu blokda joylashgan yoki shu blokdagi ichki blokda o'zgaruvchi nomi bilan bir xil bo'lmasligi kerak.

Global o'zgaruvchi mavjudlik sohasi ta'rifdan, to dastur oxirigacha bo'ladi. Agar ichki blokdagi o'zgaruvchi nomi global o'zgaruvchi nomi bilan bir xil bo'lsa, lokal o'zgaruvchi ko'rinish sohasida global o'zgaruvchiga kvalifikatsiya operatori yordamida murojaat qilish mumkin.

Misol:

```
#include<iostream>
using namespace std;
int i=5;
int main()
{
int i=9;
cout<<i<<endl;
cout<<::i;
return 0;
}
```

Natija:

```
9
5
```

Nomlar fazosi. C++ tilida nomlar fazosi (namespace) mexanizmi ilovani bir necha sohalarga ajratish imkonini beradi. Nomlar fazosini e'lon qilish uchun namespace kalit so'zidan foydalaniladi:

namespace<identifikator> {[<e'lon qilish >]}

Eng yuqori ko'rinish sohasi global nomlar fazosi deb ataladi. Global nomlar fazosiga murojaat qilish sintaksisi.

::globalNom;

Standart nomlar fazosi std deb nomlanadi va C++ standart kutubxonalariga kirgan hamma nomlarni o'z ichiga oladi.

Masalan:

```
#include <iostream>
#include <string>
using namespace std;
int main(void)
{
string name;
cout<< "What is your name my lord?" <<endl;
cin>> name;
cout<< "\nHello Sir" << name.c_str() << endl;
return 0;
}
```

Global nomlar fazosidan foydalanib, bu dasturni yozish uchun global ruxsat berish operatoridan foydalanish lozim bo'ladi:

```
#include <iostream>
#include <string>
int main(void)
{
std::string name;
std::cout<< "What is your name my lord?" <<std::endl;
std::cin>> name;
std::cout<< "\nHello Sir" << name.c_str() << std::endl;
return 0;
}
```

Shunday qilib standart nomlar fazosidan foydalanish dasturlashni yengillashtiradi.

12-bob bo'yicha savollar

1. O'zgaruvchilar qanday ta'riflanadi?
2. Asosiy turlarni ko'rsating.

3. Qanday konstantalar mavjud?
4. Amallarni ko'rsating.
5. Satr simvolli massivdan qanday farq qiladi?
6. Massivlarni initsializatsiya qilish usullarini ko'rsating.
7. Satrlarni initsializatsiya qilish usullarini ko'rsating.
8. Qanday qilib massivlar formal parametrlar sifatida ishlatilishi mumkin?
9. Turlarni keltirish qoidalarni ko'rsating.
10. Nomlar fazosi nima uchun ishlatiladi?

12-bob bo'yicha topshiriqlar

1. Berilgan eps aniqlikda umumiy hadi 1/n! bo'lgan ketma-ketlik yig'indisini hisoblovchi dastur tuzing.
2. Umumiy hadi x/n! bo'lgan ketma-ketlik n ta hadi yig'indisini hisoblovchi dastur tuzing.
3. Kiritilgan n ta son qat'iy o'suvchi ekanligini tekshiruvchi dastur yarating.
4. Kiritilgan n simvoldan nechtasi unli harf ekanligini switch operatori yordamida hisoblovchi dastur tuzing.
5. Rekursiya yordamida Paskal uchburchagini hisoblovchi funksiya tuzing. Bu funksiya yordamida uchburchakni ekranga chiqaruvchi funksiya tuzib, dasturda foydalaning.

13-bob. SINFLAR VA OBYEKTLAR

13.1. Strukturalar

Struktura – bu ma'lumotlarni bir butun nomlangan elementlar to'plamiga birlashtirish. Struktura elementlari (maydonlar) har xil turda bo'lishi mumkin va ular har xil nomlarga ega bo'lishi kerak.

Strukturali tur quyidagicha aniqlanadi:

struct {<ta'riflar ro'yxati >}

Strukturada, albatta, bitta komponenta bo'lishi kerak. Struktura turidagi o'zgaruvchi quyidagicha ta'riflanadi:

<struktura_nomi ><o'zgaruvchi>;

Struktura turidagi o'zgaruvchi ta'riflanganda initsializatsiya qilinishi mumkin:

<struktura_nomi ><o'zgaruvchi>=<initsializator>;

Strukturani initsializatsiyalash uchun uning elementlari qiymatlari figurali qavslarda tavsiflanadi.

Misollar:

1. **struct Student**
{
char name[20];
int kurs;
float rating;
};
Student s={"Qurbonov", 1, 3.5};
2. **struct**
{
char name[20];
char title[30];
float rate;
}employee={"Ashurov", "direktor", 10000};

Strukturalarni o'zlashtirish. Bitta tuzilma turdagi o'zgaruvchilar uchun o'zlashtirish operatsiyasi aniqlangan. Bunda har bir elementdan nusxa olinadi. Masalan:

```
Student ss=s;
```

Struktura elementlariga murojaat. Struktura elementlariga murojaat aniqlangan ismlar yordamida bajariladi:

```
<Struktura_nomi>.<element_nomi>
```

Masalan:

employee.name – "Ashurov" qiymatga ega bo'lgan o'zgaruvchi;

employee.rate – 10000 qiymatga ega bo'lgan butun turdagi o'zgaruvchi

Quyidagi 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>
using namespace std;
#include <math.h>
struct
{
double mass;
float coord[3];
} point={12.3, {1.0, 2.0, –3.0}};
int main()
{
int i;
float s=0.0;
for (i=0; i<3; i++)
s+=point.coord[i]*point.coord[i];
cout<<"\n masofa="<<sqrt(s); return 0;
}
```

Strukturalar va funksiyalar. Strukturalar funksiyalar argumentlari sifatida yoki funksiya qaytaruvchi qiymat sifatida kelishi mumkin. Bundan tashqari, funksiya argumenti sifatida struktura turidagi massiv kelishi mumkin.

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 yig'indisini hisoblash funksiyasi:

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

Misol:

```
#include<iostream>
using namespace std;
struct person
{
char name[20];
int year;
};

person old_person(person a[], int n)
{
int i;
person s=a[0];
for(i=1; i<n; i++)
if(a[i].year>s.year) s=a[i];
return s;
}
void print_person(person s)
{
cout<<"name="<<s.name;
cout<<"year="<<s.year<<endl;
}
int main()
{
person a[]={{"smit", 34}, {"bobbi", 45}, {"pit", 56}};
person s=old_person(a, 3);
print_person(s);
return 0;
}
```

Funksiyada bitta struktura turidagi o'zgaruvchi qiymatini o'zgartirish mumkin emas, lekin massiv elementlari qiymatini o'zgartirish mumkin:

```
#include<iostream>
using namespace std;
struct goods {
char name[20];
long price;
float percent;
};
void change_percent(goods a[], int n, float percent)
{
int i;
for(i=1; i<n; i++) a[i].percent=percent;
}
void print_goods(goods s)
{
cout<<s.name<<" "<<s.price<<" "<<s.percent<<endl;
}
void all_print(goods a[], int n)
{
int i;
for(i=0; i<n; i++) print_goods(a[i]);
};

int main()
{
goods a[]={{"smit", 34, 0.5}, {"bobbi", 45, 0.7}, {"pit", 56, 0.8}};
all_print(a, 3);
change_percent(a, 3, 0.5);
all_print(a, 3);
int ii; cin>>ii;
return 0;
}
```

13.2. Sinf ta'rifi

Sinf struktura tushunchasi kengaytmasi sifatida. Sinflarni eng sodda holda quyidagicha tasvirlash mumkin:

Sinf-kaliti Sinf-soni {komponentalar ro'yxati}

Sinf komponentalari sodda holda turlangan ma'lumotlar va funksiyalardan iborat bo'ladi. Figurali qavslarga olingan komponentalar ro'yxati sinf tanasi deb ataladi. Sinfga tegishli funksiyalar komponenta-funksiyalar yoki sinf funksiyalari deb ataladi.

Sinf kaliti sifatida struct xizmatchi so'zi ishlatilishi mumkin. Masalan, quyidagi konstruksiya kompleks son sinfini kiritadi.

```
struct complex
{
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 funksiya (define() va display()) kiritilgan.

Bu kiritilgan sinf o'zgaruvchilar turi deb qaralishi mumkin. Bu turlar yordamida konkret obyektlarni quyidagicha tasvirlash mumkin:

Misol uchun:

```
complex x, y;
complex dim[8];
```

Sinfga tegishli obyektlar quyidagicha tasvirlanadi;

Sinf-nomi.obyekt-nomi

Dasturda obyekt komponentasiga quyidagicha murojaat qilish mumkin:

Sinf-nomi.obyekt-nomi:: komponenta-nomi yoki soddaroq holda

Obyekt-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 funksiyalarga quyidagicha murojaat qilinadi:

obyekt-nomi.funksiya-nomi

Misol uchun:

x. define(0.9) (Bu holda real=0.9 va imag=0.0)

x. define(4.3,20.0) (Bu holda kompleks son 4.3+i*20.0)

display funksiyasi ekranda kompleks son qiymatlarini tasvirlaydi.

Komponenta o'zgaruvchilar va funksiyalar. Sinf komponenta o'zgaruvchilari sifatida o'zgaruvchilar, massivlar, ko'rsatkichlar ishlatilishi mumkin. Elementlar ta'riflanganda initsializatsiya qilish mumkin emas. Buning sababi shuki, sinf uchun xotiradan joy ajratilmaydi. Komponenta elementlariga komponenta funksiyalar orqali murojaat qilinganda, faqat nomlari ishlatiladi. Sinfdan tashqarida sinf elementlariga emas obyekt elementlariga murojaat qilish mumkin. Bu murojaat ikki xil bo'lishi mumkin.

Obyekt-nomi.Element-nomi.

Sinf elementlari sinfga tegishli funksiyalarida ishlatilishidan oldin ta'riflangan bo'lishi shart emas. Xuddi shunday bir funksiyadan hali ta'rifi berilmagan ikkinchi funksiyaga murojaat qilish mumkin.

Komponentalarga murojaat huquqlari. Komponentalarga murojaat huquqi murojaat spetsifikatorlari yordamida boshqariladi. Bu spetsifikatorlar:

protected – himoyalangan;

private – xususiy;

public – umumiy.

Himoyalangan komponentalardan sinflar ierarxiyasi qurilganda foydalaniladi. Oddiy holda protected spetsifikatori private spetsifikatoriga ekvivalentdir. Umumiy, ya'ni public turidagi komponentalarga dasturning ixtiyoriy joyida murojaat qilinishi mumkin.

Xususiy, ya'ni private turidagi komponentalarga sinf tashqarisidan murojaat qilish mumkin emas. Agar sinflar struct xizmatchi 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 xizmatchi so'zi orqali ta'riflangan bo'lsa, uning hamma komponentalari xususiy bo'ladi. Lekin bu huquqni murojaat spetsifikatorlari yordamida o'zgartirish mumkin.

Bu spetsifikator yordamida sinflar umumiy holda quyidagicha ta'riflanadi:

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

Sinf ta'riflangandan so'ng, shu sinf turidagi o'zgaruvchilarni (obyektlarni) quyidagicha ta'riflash mumkin:

class_name object_one, object_two, object_three;

Quyidagi misolda employee, sinfi kiritilgan:

```
class employee
{
public:
long employee_id;
float salary;
void show_employee(void)
{
cout<<"Nomer:" <<employee_id<<endl;
cout<<"Maosh:" <<salary<<endl;
};
};
```

Bu sinf ikki o'zgaruvchi va bitta funksiya-elementga ega.

Quyidagi dastur ikki employee obyektini yaratadi. Nuqta operatordan foydalanib, ma'lumot elementlarga qiymat beriladi, so'ngra **show_employee** elementidan foydalanib, xizmatchi haqidagi ma'lumot ekranga chiqariladi:

```
#include <iostream>
using namespace std;
class employee
{
public:
long employee_id;
float salary;
void show_employee(void)
```

```
{
cout<<"Nomer:" <<employee_id<<endl;
cout<<"Maosh:" <<salary<<endl;
};
};
int main()
{
employee worker, boss;
worker.employee_id=12345;
worker.salary=25000;
boss.employee_id=101;
boss.salary=101101.00;
cout<<"\n"<<"ishchi"<<endl;
worker.show_employee();
cout<<"\n"<<"boss"<<endl;
boss.show_employee();
return 0;
}
```

13.3. Sinf komponenta funksiyalari

Komponenta funksiya ta'rifi. Komponenta funksiya, albatta, sinf tanasida ta'riflangan bo'lishi lozim. Global funksiyalardan farqli komponenta funksiya sinfning hamma komponentalariga murojaat qilishi mumkin. Funksiyaning faqat prototipi emas, to'la ta'rifi sinf tanasida joylashgan bo'lsa, bu funksiya joylashtiruvchi (inline) funksiya hisoblanadi. Ma'lumki, inline funksiyalarda sikllar, kalit bo'yicha o'tish operatori ishlatilishi mumkin emas. Bundan tashqari bunday funksiyalar rekursiv funksiya bo'la olmaydi. Bu chegaralarni yengish uchun sinf tanasiga faqat funksiya prototipi joylashtirilib, funksiyaning to'la ta'rifi sinf tashqarisida dasturga kiruvchi boshqa funksiyalar bilan birga beriladi. Komponenta funksiya sinf tashqarisida ta'riflanganda, qaysi sinfga tegishli ekanligini quyidagi shaklda ko'rsatiladi:

Sinf-nomi:: Komponenta funksiya-nomi

Sinf tanasiga komponenta funksiya prototipi quyidagi shaklda joylashtiriladi:

Tur funksiya-nomi (formal-parametrlar-ta'rifi)

Sinf tashqarisida funksiya quyidagi shaklda ta'riflanadi:

Tur sinf-nomi:: funksiya-nomi (formal-parametrlar-spetsifikatsiyasi)
{funksiya tanasi};

Oldingi misoldagi **employee** sinfida funksiya sinf ichida ta'riflangan. Bunday funksiya joylanuvchi **(inline) funksiya** deb qaraladi.

Funksiyani sinf tashqarisida ta'riflab, sinf ichiga funksiya prototipini joylashtirish mumkin. Sinf ta'rifi bu holda quyidagi ko'rinishda bo'ladi:

```
class employee
{
public:
long employee_id;
float salary;
void show_employee(void);
};
```

Har xil funksiyalar bir xil nomli funksiyalardan foydalanishi mumkin bo'lgani uchun funksiya nomi, sinf nomi va global ruxsat operatori belgisi (::) qo'yilishi lozim.

```
void employee::show_employee(void)
{
cout<<"Nomer:" <<employee_id<<endl;
cout<<"Maosh:" <<salary<<endl;
};
```

Funksiya sinf tashqarisida ta'riflangan bo'lsa, ularni inline funksiya sifatida qarash uchun funksiya ta'rifida inline so'zi aniq ko'rsatilgan bo'lishi kerak.

Quyidagi dastur **show_employee** funksiyasi ta'rifini sinf tashqarisiga joylashtiradi va inline so'zi aniq ko'rsatiladi:

```
#include <iostream>
using namespace std;
class employee
{
public:
long employee_id;
float salary;
void show_employee(void);
};
```

```
inline void employee::show_employee(void)
{
cout<<"Nomer:" <<employee_id<<endl;
cout<<"Maosh:" <<salary<<endl;
};

int main()
{
employee worker, boss;
worker.employee_id=12345;
worker.salary=25000;
boss.employee_id=101;
boss.salary=101101.00;
cout<<"\n"<<"ishchi"<<endl;
worker.show_employee();
cout<<"\n"<<"boss"<<endl;
boss.show_employee();
return 0;
}
```

13.4. Konstruktor va destruktor

Konstruktorlar. Konstruktorlar bu sinf komponenta funksiyalari bo'lib, obyektlarni avtomatik initsializatsiya qilish uchun ishlatiladi.

Konstruktorlarning ko'rinishi quyidagicha bo'lishi mumkin:

Sinf nomi (formal parametrlar ro'yxati)
{konstruktor tanasi}

Bu komponenta funksiya nomi sinf nomi bilan bir xil bo'lishi lozim.

Misol uchun complex sinfi uchun konstruktorni quyidagicha kiritish mumkin:

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

Konstruktorlar uchun qaytariluvchi turlar, hatto void turi ham ko'rsatilmaydi. Dasturchi tomonidan ko'rsatilmagan holda ham obyekt yaratilganda konstruktor avtomatik ravishda chaqiriladi.

Masalan, obyekt complex cc; shaklida aniqlangan bo'lsa, konstruktor avtomatik chaqirilib, real va imag parametrlari avtomatik ravishda 0.0 qiymatlariga ega bo'ladi.

Ko'zda tutilgan holda parametrsiz konstruktor va quyidagi turdagi nusxa olish konstruktorlari yaratiladi: T:: T (const T&)

Misol uchun

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

Sinfda bir nechta konstruktorlar bo'lishi mumkin, lekin ularning faqat bittasida parametrlar qiymatlari oldindan ko'rsatilgan bo'lishi kerak.

Konstruktor adresini hisoblash mumkin emas. Konstruktor parametri sifatida o'z sinfining nomini ishlatish mumkin emas, lekin bu nomga ko'rsatkichdan foydalanish mumkin.

Konstruktorni oddiy komponenta funksiya sifatida chaqirib bo'lmaydi. Konstruktorni ikki xil shaklda chaqirish mumkin:

Sinf_nomi .Obyekt_nomi (konstruktor_haqiqiy_parametrlari)

Sinf_nomi (konstruktor_haqiqiy_parametrlari)

Birinchi shakl ishlatilganda haqiqiy parametrlar ro'yxati bo'sh bo'lmasligi lozim. Bu shakldan yangi obyekt 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() // xato
```

Konstruktorni ikkinchi shaklda chaqirish nomsiz obyekt yaratilishiga olib keladi. Bu nomsiz obyektdan ifodalarda foydalanish mumkin.

Misol uchun:

complex ZZ=complex (4.0;5.0);

Bu ta'rif orqali ZZ obyekt yaratilib, unga nomsiz obyekt qiymatlari (real=4.0; imag=5.0) beriladi;

Konstruktor nomi sinf nomi bilan bir xil bo'lishi lozim. Misol uchun siz employee sinfdan foydalansangiz, konstruktor ham employee nomga

ega bo'ladi. Agar dasturda konstruktor ta'rifi berilgan bo'lsa, obyekt yaratilganda avtomatik chaqiriladi. Quyidagi dasturda employee nomli sinf kiritilgan:

```
class employee
{
public:
employee(long, float);
void show_employee(void);
private:
long employee_id;
float salary;
};
```

Konstruktor ta'rifi:

```
employee::employee(long empl_id, float sal)
{
employee_id=empl_id;
if (salary<50000.0)
salary=sal;
else
salary=0.0;
}
```

Shu sinfdan foydalanilgan dastur:

```
#include <iostream>
using namespace std;
class employee
{
public:
employee(long, float);
void show_employee(void);
private:
long employee_id;
float salary;
};
employee::employee(long empl_id, float sal)
```



```
{
employee_id=empl_id;
if (salary<50000.0)
salary=sal;
else
salary=0.0;
}
void employee::show_employee(void)
{
cout << "Nomer:" << employee_id << endl;
cout << "Maosh:" << salary << endl;
}
int main()
{
employee worker(101, 10101.0);
cout<<"ishchi"<<endl;
worker.show_employee();
return 0;
}
```

Konstruktordan foydalanib, obyekt ta'riflanganda parametr uzatish mumkin: employee worker(101, 10101.0);

Agar dasturda employee turidagi obyektlar mavjud bo'lsa, har birini quyidagicha initsializatsiya qilish mumkin

employee worker(101, 10101.0);
employee secretary(57, 20000.0);
employee manager(1022, 30000.0);

Satrli maydonga misol. Keyingi misolda satrli maydon o'zgaruvchi sifatida beriladi.

```
#include <iostream>
#include <string.h>
using namespace std;
class employee
{
char name[20];
long employee_id;
float salary;
```

```
cout << "Maosh:" << salary << endl;
}
int main()
{
cout<<"ishchi"<<endl;
employee worker(101, 10101.0);
worker.show_employee();
cout<<"manager"<<endl;
employee manager(102);
manager.show_employee();
return 0;
}
```

Obyektlar massivlari. Obyektlar massivi ta'riflash uchun sinf ko'zda tutilgan (parametrsiz) konstruktorga ega bo'lishi kerak.

Obyektlar massivi ko'zda tutilgan konstruktor tomonidan yoki har bir element uchun konstruktor chaqirish yo'li bilan initsializatsiya qilinishi mumkin.

class complex a[20]; //ko'zda tutilgan parametrsiz konstruktorni chaqirish

class complex b[2] = {complex(10),complex (100)}; //oshkor chaqirish

Quyidagi misolda **player** sinfi kiritiladi. Dasturda sinf funksiyasi **show_player** va konstruktor tashqarisida ta'riflanadi. So'ngra **player** turidagi ikki massiv yaratilib, har biri haqidagi ma'lumot ekranga chiqariladi

```
#include <iostream>
#include <string>
using namespace std;
class player
{
public:
player();
player (string name, int weight, int age);
void show_player (void);
private:
string name;
int weight;
int age;
```



```
};
player::player()
{
name=" ";
weight=0;
age=0;
};
player::player(string name, int weight, int age)
{
player::name=name;
player::weight=weight;
player::age=age;
};
void player::show_player (void)
{
cout<<"Ism:" << name << endl;
cout<<"Vazn:" << weight << endl;
cout<<"Yosh:" << age << endl;
}
class array_player
{public:
void show_array(player a[], int n)
{for(int i=0; i<n; i++)
{a[i].show_player();cout<<endl;}}
void input_array(player a[], int n)
{string name; int weight, age;
for(int i=0; i<n; i++)
{cin>>name>>weight>>age;
a[i]=player(name, weight, age);
}
}
};
int main()
{array_player arr;
Player happy[]={player("Olimov", 58, 24),
player("Alimov", 72, 35)};
arr.show_array(happy, 2);
```

```
static int page_count;
float price;
};
int book_series::page_count;
void book_series::set_pages(int pages)
{
page_count=pages;
}
book_series::book_series(float price)
{
book_series::price=price;
}
void book_series:: show_book (void)
{
cout << "Narx:" << price << endl;
cout << "Betlar:" << page_count << endl;
}
int main()
{
book_series programming(213.95);
book_series word(19.95);
word.set_pages(256);
programming.show_book ();
word.show_book();
cout << endl << «page_count ning o'zgarishi "<< endl;
programming.set_pages(512);
programming.show_book();
word.show_book();
return 0;
}
```

Ko'rinib turganidek, sinf **page_count** ni **static int** sifatida e'lon qiladi. Sinfni aniqlagandan so'ng, dastur shu vaqtning o'zida page_count elementini global o'zgaruvchi sifatida e'lon qiladi. Dastur page_count elementini o'zgartirganda, o'zgarish shu vaqtning o'zidayoq book_series sinfining barcha obyektlarida namoyon bo'ladi.

Agar obyektlar mavjud bo'lmasa, public static atributli elementlardan foydalanish. Sinf elementini statik kabi e'lon qilishda bu element ushbu sinfning barcha obyektlari tomonidan birgalikda qo'llanadi. Biroq shunday vaziyatlar yuz berishi mumkinki, dastur hali obyektni yaratganicha yo'q, ammo u elementdan foydalanishi kerak. Elementdan foydalanish uchun dastur uni public va static sifatida e'lon qilishi kerak. Masalan, quyidagi dasturda, hatto book_series sinfidagi obyektlar mavjud bo'lmasa ham, bu sinfning page_count elementidan foydalaniladi:

```
#include <iostream>
using namespace std;
class book_series
{
public:
static int page_count;
private:
float price;
};
int book_series::page_count;
int main()
{
book_series::page_count=256;
cout << "page_count ning joriy qiymati "<< book_series::page_
count <<"ga teng" <<endl;
return 0;
}
```

Bu o'rinda, sinf page_count elementini public sifatida e'lon qilgani uchun, hatto agar book_series sinfidagi obyektlar mavjud bo'lmasa ham dastur sinfning ushbu elementiga murojaat qilishi mumkin.

Statik funksiya-elementlardan foydalanish. Avvalgi dastur ma'lumotlar **statik** elementlarining qo'llanishini ko'rsatib bergan edi. C++ xuddi shunday usul bilan **statik** funksiya-elementlar (usullar)ni aniqlash imkonini beradi. Agar **statik** usul yaratilayotgan bo'lsa, dastur bunday usulni, hatto uning obyektlari yaratilmagan holda ham chaqirib olishi mumkin. Masalan, agar sinf sinfdan tashqari ma'lumotlar uchun qo'llanishi mumkin bo'lgan usulga ega bo'lsa, siz bu usulni statik qila olishingiz mumkin bo'lardi. Funksiyadan foydalanish uchun dastur uni public va static sifatida

e'lon qilishi kerak. Masalan, quyidagi dasturda, hatto book_series sinfidagi obyektlar mavjud bo'lmasa ham, bu sinfning show_count() usulidan foydalaniladi:

```
#include <iostream>
using namespace std;
class book_series
{
public:
static int show_count() {return page_count;};
private:
float price;
static int page_count;
};
int book_series::page_count=256;
int main()
{
cout << "page_count ning joriy qiymati" << book_series::show_
count() <<"ga teng" << endl;

return 0;
}
```

14.2. Satr sinf sifatida

String turi. Satrlar bilan ishlash uchun standart kutubxonaga kiruvchi string sinfidan foydalanish qulay.

Bu turdan foydalanish uchun quyidagi sarlavhali faylni ulash lozim:

```
#include <string.n>
```

Satrlarni ta'riflashga misollar:

```
string st("BAHO \n"); //simvollar satri bilan initsiallash
string st2; // bo'sh satr
string st3(st); shu turdagi o'zgaruvchi bilan initsiallash
```

Satrlar ustida amallar. Satrlar ustida quyidagi amallar aniqlangan:

- qiymat berish (=);
- ikki amal ekvivalentligini tekshirish uchun (==) va (!=);
- konkatenatsiya yoki satrlarni ulash (+);
- qiymat berib qo'shish amali (+=);

- indeks olish ([]).

Usullar. Satr uzunligini aniqlash uchun **size()** funksiyasidan foydalaniladi (uzunlik tugallovchi simvolni hisobga olmaydi).

```
cout << "uzunlik" << st << ":" << st.size();
```

Maxsus **empty()** usuli agar satr bo'sh bo'lsa true qaytaradi, aks holda false qaytaradi:

```
if (st.empty()) // to'g'ri: bo'sh
```

Misol:

```
#include<iostream>
using namespace std;
int main() {
string str("fa.disney.com");
int size=str.size();
for (int ix=0; ix<size; ++ix) if (str[ ix ]== '.') str[ ix ]='_';
return 0;
}
```

Misol. Struktura ta'rifi va initsializatsiyasida **string** turidan foydalanish.

```
#include<iostream>
using namespace std;
struct Student
{
string name;
int kurs;
float rating;
};
void Student_show(Student a)
{
cout<<name<<endl;
cout<<kurs<<endl;
cout<<rating<<endl;
}
int main() {
Student s={"Mahmudov", 1, 3.5};
Student_show(s);
return 0;
```

}

Satrli maydonga misol. Keyingi misolda satrli maydon string turidagi o'zgaruvchi sifatida beriladi.

```
#include <iostream>
#include <string>

using namespace std;

class employee
{
public:
employee(string, long, float);
void show_employee(void);
int change_salary(float);
long get_id(void);
private:
string name;
long employee_id;
float salary;
};
employee::employee(string name, long employee_id, float salary)
{
employee::name=name;
employee::employee_id=employee_id;
if (salary<50000.0)
employee::salary=salary;
else
employee::salary=0.0;
}
void employee::show_employee(void)
{
cout << "Ism:" << name << endl;
cout << "Nomer:" << employee_id << endl;
cout << "Maosh:" << salary << endl;
}
int main()
{
```



```
employee worker("Happy Jamsa", 101, 10101.0);
worker.show_employee();
return 0;
}
```

Natija:
Ism: Happy Jamsa
Nomer: 101
Maosh: 10101

14.3. Sinflar do'stlari

Bir sinf ikkinchi sinfning umumiy usullaridan foydalanishi mumkin. Masalan, bir sinf obyektlari massivlari bilan ishlovchi funksiyalarni ikkinchi sinf oshkor statik usullari sifatida ta'riflash mumkin.

```
#include <iostream>
using namespace std;
class goods
{
string name;
long price;
float percent;
public:
goods(){};
goods(string name, long price, float percent)
{
goods::name=name;
goods::price=price;
goods::percent=percent;
}
void print()
{
cout<<name<<endl;
cout<<price;
cout<<percent;
};
};
```

```
class array_goods
{
public:
static void all_input(struct goods a[], int n)
{
string name;
long price;
float percent;
for(int i=0; i<n; i++) {
cin>>name>>price>>percent;
a[i]=goods(name, price, percent);
}
};
static void all_print(struct goods a[], int n)
{
for(int i=0; i<n; i++) a[i].print();
};
};

int main()
{
goods a[]={goods("smit", 34, 0.5), goods("bobbi", 45, 0.7),
goods("pit", 56, 0.8)};
array_goods::all_print(a, 3);
array_goods::all_input(a, 3);
array_goods::all_print(a, 3);
return 0;
}
```

Sinf do'stlari ta'rifi. Sinfning komponentalariga murojaat qilishning yana bir usuli dostona funksiyalardan foydalanishdir. Sinfning dostona funksiyasi deb shu sinfga tegishli bo'lmagan, lekin shu sinfning himoyalangan komponentlariga murojaat qilish huquqiga ega bo'lgan funksiyalarga aytiladi. Funksiya dostona bo'lishi uchun sinf tanasida friend spetsifikatori bilan ta'riflanishi lozim.

Do'stona funksiyani ta'riflash:

friend <funksiya prototipi >

Do'stona funksiyalardan foydalanish xususiyatlari quyidagilar:

Do'stona funksiya murojaat qilinganda this ko'rsatkichiga ega bo'lmaydi.

Sinf obyektlari dostona funksiyaga parametrlari orqali uzatilishi lozim.

Do'stona funksiya sinf komponentasi bo'lmagani uchun unga tanlov amalini qo'llab bo'lmaydi:

Sinf obyekti.funksiya nomi va Obyektga_ko'rsatkich-funksiya nomi.

Do'stona funksiyaga murojaat spetsifikatorlari (public, protected, private) qo'llanmaydi.

Do'stona funksiya prototipining sinf usulida joylashtirilishining farqi yo'q.

Do'stona funksiyalar mexanizmi sinflar orasidagi aloqani soddalashtirishga imkon beradi. Sinflardan berkitilgan komponentalariga murojaat qilish uchungina kiritilgan funksiyalarni olib tashlash mumkin.

Misol tariqasida «sohadagi nuqta» va «sohadagi chiziq» sinflari uchun dostona funksiyani qarab chiqamiz. Sohadagi nuqta sinfiga, (x, y) koordinatalarini aniqlovchi komponentalar kiradi. Sohadagi chiziq sinfining komponentalari chiziqning umumiy tenglamasi $A*x+B*y+C=0$ koeffitsiyentlari A, B, C.

Quyidagi dasturda ikkala sinf uchun dostona bo'lgan nuqtadan chiziqqacha masofani hisoblashga imkon beradigan funksiya kiritilgan.

```
#include <iostream.h>
class line;
class point
{
float x, y;
public:
point(float xn=0, float yn=0)
{
x=xn; y=yn;
}
friend float masofa(point, line);
};
class line
{
float A, B, C;
```

```
public:
line(float a, float b, float c)
{
A=a; B=b; C=c;
}
friend float masofa(point, line);
};
float masofa(point P, line L)
{
return L.A*P.x+L.B*P.y+L.C;
};

int main()
{
point P(16.0, 12.3);
line L(10.0, –42.3, 24.0);
cout << "\n P nuqtasi L chiziqdan cheklanishi:";
cout << masofa(P, L);
return 0;
}
```

Dastur bajarilishi natijasi

P nuqtasi L chiziqdan cheklanishi: –336.29009

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

Do'stona sinflarni ta'riflash:

friend <sinf nomi>

Quyidagi misolda book sinfi librarian sinfini o'ziga dostona sinf deb belgilagan:

```
class book
{
public:
book(string, string, string);
void show_book(void);
friend librarian;
private:
```



```
string title;
string author;
string catalog;
};
```

Shuning uchun librarian sinf obyektlari book sinfning xususiy elementlariga, nuqta operatoridan foydalangan holda, to'g'ridan to'g'ri murojaat etishi mumkin:

```
#include <iostream>
#include <string>
using namespace std;
class librarian;
class book
{
public:
book(string, string, string);
void show_book(void);
friend librarian;
private:
string title;
string author;
string catalog;
};
book::book(string title, string author, string catalog)
{
book::title=title;
book::author=author;
book::catalog=catalog;
}
void book::show_book(void)
{
cout << "Nomi:" << title << endl;
cout << "Muallif:" << author << endl;
cout << "Katalog:" << catalog << endl;
}
class librarian
```

```
{
public:
void change_catalog(book &, string);
string get_catalog(book);
};
void librarian::change_catalog(book& this_book, string new_
catalog)
{
this_book.catalog=new_catalog;
}
string librarian::get_catalog(book this_book)
{
string temp_catalog;
temp_catalog=this_book.catalog;
return(temp_catalog);
}
int main()
{
book programming ("C++ tilida dasturlashni o'rganamiz",
"Jamsa", "P101");
librarian library;
programming.show_book();
library.change_catalog(programming, "Engil C++ 101");
programming.show_book();

return 0;
}
```

Ko'rib turganimizdek, dastur librarian sinfining change_catalog funksiyasiga book obyektini adres orqali bermoqda. Bu funksiya sinfning book elementini o'zgartirgani uchun, dastur parametrni adres orqali uzatishi va undan so'ng ushbu sinf elementiga murojaat uchun ko'rsatkich ishlatmog'i lozim. Misolda book sinfi aniqlanishidan friend operatori o'chirib yuborilsa, C++ kompilyatori har gal book sinfi xususiy ma'lumotlariga murojaatda sintaksik xato haqida xabar chiqaradi

Do'stlar sonini chegaralash. Agarda bir nechta sinf funksiyalariga boshqa sinfning xususiy ma'lumotlariga murojaat qilish kerak bo'lsa, u

holda C++ dostona sinfning faqatgina belgilangan funksiyalari xususiy elementlarga murojaat etishiga imkoniyat beradi. Masalan, faqatgina change_catalog va get_catalog funksiyalarga book sinfning xususiy elementlariga murojaat kerak. Quyida ko'rsatilganidek, book sinfning ichida faqatgina shu funksiyalarda xususiy funksiyalarga murojaat chegarasini qo'yishi lozim:

```
class book
{
public:
book(string, string, string);
void show_book (void);
friend string librarian::get_catalog(book);
friend void librarian::change_catalog(book&, string);
private:
string title;
string author;
string catalog;
};
```

Ko'rib turganimizdek, friend operatorlari xususiy elementlarga murojaat qiluvchi hamma do'st funksiyalarining to'liq prototiplarini o'z ichiga oladi.

Agar dastur bir sinfdan boshqasiga murojaat qilsa va sinflar aniqlanish tartibi noto'g'ri bo'lsa, sintaksis xatoga duch kelish mumkin. Bizning holda book sinfi librarian sinfida e'lon qilingan funksiyalar prototiplariga murojaat qilmoqda. Shuning uchun librarian sinfi aniqlanishi book sinfi aniqlanishidan oldin kelishi kerak, biroq librarian sinfi book sinfiga murojaat qilmoqda:

```
class librarian
{
public:
void change_catalog(book&, string);
string get_catalog(book);
};
```

Misolda book sinfi aniqlanishini librarian sinfi aniqlanishidan oldin qo'yib bo'lmagani uchun C++ book sinfini e'lon qilish imkonini beradi va

shu bilan u kompilatorga bunday sinf borligi haqida xabar beradi va keyinroq o'zi ham aniqlanadi. Quyida buni qanday amalga oshirish keltirilgan:

```
class book; // sinf elon kilinishi
```

Quyidagi dasturda librarian sinfining ayrim usullariga book sinfining xususiy elementlariga murojaat qilish imkoniyati berilgan. Sinflar tartibiga ahamiyat bering:

```
#include <iostream>
#include <string>

using namespace std;
class book;
class librarian
{
public:
void change_catalog(book&, string);
string get_catalog(book);
};
class book
{
public:
book(string, string, string);
void show_book (void);
friend string librarian::get_catalog(book);
friend void librarian::change_catalog(book&, string);
private:
string title;
string author;
string catalog;
};
book::book(string title, string author, string catalog)
{
book::title=title;
book::author=author;
book::catalog=catalog;
}
void book::show_book(void)
```



```
{
cout << "Nomi:" << title << endl;
cout << "Muallif:" << author << endl;
cout << "Katalog:" << catalog << endl;
}
void librarian::change_catalog(book& this_book, string new_
catalog)
{
this_book.catalog=new_catalog;
}
string librarian::get_catalog(book this_book)
{
string temp_catalog;
temp_catalog=this_book.catalog;
return(temp_catalog);
}
int main()
{
book programming("C++ tilida dasturlashni o'rganamiz", "Jam-
sa", "P101");
librarian library;
programming.show_book();
library.change_catalog(programming, "Engil C++ 101");
programming.show_book();
return 0;
}
```

14.4. Sinflarning boshqa sinflardan tashkil topishi

Obyekt maydon sifatida. Murakkab sinflarni hosil qilishda oldin uni tashkil etuvchi oddiyroq sinflarni e'lon qilib, keyin esa ularni birlashtirish orqali sinfni hosil qilish maqsadga muvofiq. Masalan, g'ildirak sinfi, motor sinfi, uzatish korobkasi sinfi va boshqa sinflarni hosil qilib, keyin esa ularni birlashtirish orqali avtomobil sinfini qurish oldimizga turgan masalani yechishni ancha osonlashtiradi.

Ikkinchi misolni ko'rib chiqamiz. To'g'ri to'rtburchak chiziqlardan tashkil topgan. Chiziq esa ikki nuqta orqali aniqlanadi. Har bir nuqta x va

y koordinatalar yordamida aniqlanadi. Quyidagi dasturda to'rtburchak sinfi ko'rsatilgan. To'g'ri to'rtburchak diagonal bo'yicha ikki nuqta va ikki tomon yordamida aniqlanadi. Shuning uchun oldin har bir nuqtaning x va y koordinatalarini saqlash uchun nuqta sinfi e'lon qilingan.

Nuqta va to'g'ri to'rtburchakning e'lon qilinishi

```
#include <iostream>
using namespace std;
class Point
{
public:
Point(int x1=0, int y1=0)
{
x=x1; y=y1;
}
int GetX() const {return x;}
int GetY() const {return y;}
private:
int x;
int y;
};

class Rectangle
{
public:
Rectangle(int x1, int y1, int x2, int y2):
p1(x1, y1), p2(x2, y2)
{
a=x2-x1;
b=y2-y1;
};
Rectangle(Point a1, Point a2):p1(a1), p2(a2)
{
a=a2.GetX() – a1.GetX();
b=a2.GetY() – a1.GetY();
};
int Per() {return 2*(a+b);}
```



```
int Sq() {return a*b;}
private:
Point p1, p2;
int a, b;
};

int main()
{
Rectangle X(10, 20, 50, 80);
cout << "Perimetr=" <<X.Per()<<endl;
cout << "Yuza=" <<X.Sq()<<endl;
cout<<endl;
Point a(2,4); Point b(5,6);
Rectangle Y(a, b);
cout << "Perimetr=" <<Y.Per()<<endl;
cout<<"Yuza=" <<Y.Sq();
return 0;
}
```

Natija:
Perimetr=200
Yuza=2400
Perimetr=10
Yuza=6

Lokal sinflar. Sinf blok ichida, masalan, funksiya tomonida ta'riflanishi mumkin. Bunday sinf lokal sinf deb ataladi. Lokal sinf komponentalariga shu sinf tariflangan blok yoki funksiya tashqarisida murojaat qilish mumkin emas. Lokal sinf statik komponentlarga ega bo'lishi mumkin emas. Lokal sinf ichida shu sinf aniqlangan soniga tegishli nomlari; statik (static) o'zgaruvchilar; tashqi (extern) o'zgaruvchilar va tashqi funksiyalardan foydalanish mumkin. Avtomatik xotira turiga tegishli o'zgaruvchilardan foydalanish mumkin emas. Lokal sinflar komponent funksiyalari faqat joylashinuvchi (inline) funksiya bo'lishi mumkin.

Quyidagi misolda moddiy nuqta sinfi yaratilib, uning ichida nuqta sinfiga ta'rif berilgan va nuqta sinfi obyekti maydon sifatida kelgan:

```
#include <iostream>
```

```
using namespace std;
class FPoint
{
public:
//Nuqta sinfi
class Point
{
public:
Point(int x1=0, int y1=0)
{
x=x1; y=y1;
}
int GetX() const {return x;}
int GetY() const {return y;}
private:
int x;
int y;
};
//

FPoint(int x1, int y1, double w1):p(x1, y1), w(w1){};
void show()
{
cout<<"koordinata x="<<p.GetX()<<endl;
cout<<"koordinata y="<<p.GetY()<<endl;
cout<<"massa w="<<w;
}
private:
Point p;
double w;
};

int main()
{
cout<<"fizik nuqta"<<endl;
FPoint X(1, 2, 5.5);
X.show();
```



```
cout<<"\n\noddiy nuqta"<<endl;
FPoint::Point Y(2,3);
cout<<"koordinata x="<<Y.GetX()<<endl;
cout<<"koordinata y="<<Y.GetY()<<endl;
return 0;
}
```

Natija:

```
fizik nuqta
koordinata x=1
koordinata y=2
massa w=5.5

oddiy nuqta
koordinata x=2
koordinata y=3
```

Oddiy nuqta sinfi moddiy nuqta sinfining umumiy seksiyasiga joylashtirilgan.

Dasturda moddiy nuqta sinfi obyektidan tashqari nuqta sinfi obyekti ham yaratilgan. Lekin bu sinf nomi oldida moddiy nuqta nomi va kvalifikatsiya operatori joylashtirilgan.

Keyingi misolimizda nuqta sinfi ta'rifi moddiy nuqta sinfining xususiy elementlari seksiyasiga joylashtirilgan:

```
#include <iostream>
using namespace std;
class FPoint
{
public:

FPoint(int x1, int y1, double w1):p(x1, y1), w(w1){};
void show()
{
cout<<"koordinata x="<<p.x<<endl;
cout<<"koordinata y="<<p.y<<endl;
cout<<"massa w="<<w;
}
private:
```

```
//Nuqta sinfi
class Point
{
public:
Point(int x1=0, int y1=0)
{
x=x1; y=y1;
}
int x;
int y;
};
//
Point p;
double w;
};
int main()
{
cout<<"fizik nuqta"<<endl;
FPoint X(1, 2, 5.5);
X.show();
int kk; cin>>kk;
return 0;
}
```

Natija:
fizik nuqta
koordinata x=1
koordinata y=2
massa w=5.5

Bu misolda nuqta sinfining hamma elementlari umumiy, lekin dasturda bu sinfdan foydalanib bo'lmaydi.

14-bob bo'yicha savollar

1. Do'stona funksiyaning aniqlanish shaklini ko'rsating.
2. Do'stona sinflarning aniqlanish shaklini ko'rsating.

3. Sinf do'stlaridan nima uchun foydalaniladi?
4. Do'stona sinfni e'lon qilish sinfning qaysi seksiyasida ekanligi ahamiyatga egami?
5. Sinflar qanday qilib boshqa sinflardan tashkil topishi mumkin?
6. Agar bir sinf obyekti boshqa sinf maydoni bo'lsa, birinchi sinf maydonlariga qanday murojaat qilinadi?
7. Agar bir sinf obyekti boshqa sinf maydoni bo'lsa, birinchi sinf konstruktori qanday chaqiriladi?
8. Lokal sinf deb qanday sinfga aytiladi?
9. Lokal sinflar obyektlari qanday aniqlanadi?
10. Lokal sinf komponentalariga shu sinf ta'riflangan blok yoki funksiya tashqarisida murojaat qilish mumkinmi?

14-bob bo'yicha topshiriqlar

1. REKORD sinfini yarating. Bu sinfga dostona sinf yarating. Do'stona sinf usullari ma'lum shartga mos obyektlarni chiqarsin. Dasturda statik obyektlar massivi yarating. Do'stona sinf obyekti yordamida shartga mos massiv elementlarini chiqaring.
2. AVTOBUS sinfini yarating. Bu sinfga dostona sinf yarating. Do'stona sinf usullari ma'lum shartga mos obyektlarni chiqarsin. Dasturda statik obyektlar massivi yarating. Do'stona sinf obyekti yordamida shartga mos massiv elementlarini chiqaring.
3. SANA sinfini yarating. Bu sinf obyektidan maydon sifatida foydalanib, MASHG'ULOT sinfini yarating
4. Nuqta sinfini yarating. Bu sinf asosida uchburchak sinfini yarating va dasturda qo'llang. Bu sinfda perimetr, yuzani hisoblash va uchburchakni chizish usullari mavjud bo'lsin.
5. Yuqorida ko'rsatilgan misolda nuqta sinfini uchburchak sinfi ichida lokal sinf sifatida joylashtiring.

15-bob. VORISLIK

15.1. Sodda vorislik

Hosila sinflarni e'lon qilish. C++ tili ozining barcha ajdodlarining xususiyatlari, ma'lumotlari, usullari va voqealarini meros qilib oladigan hosila sinfini e'lon qilish imkoniyatini beradi. Hosila sinfda, shuningdek, yangi tavsiflarni e'lon qilish hamda meros sifatida olinayotgan ayrim funksiyalarni qo'shimcha yuklash mumkin.

Vorislik asos sinf kodidan hosila sinf nusxalarida takroran foydalanish imkonini beradi. Takroran qo'llash konsepsiyasi jonli tabiatda o'z paralleliga ega: DNK ni asos material sifatida olib qarash mumkinki, u har bir yaratilgan mavjudotdan o'zining shaxsiy turini qayta ishlab chiqish uchun takroran foydalanadi.

Hosila sinfni e'lon qilishning umumlashgan sintaksisini ko'rib chiqamiz. Seksiyalarni sanab o'tish tartibi himoya imtiyozlarini eng cheklanganlaridan, to eng ommaviylariga qarab kengayib borishiga mos keladi:

class className: [<kirish huquqini beruvchi spetsifikator>] parent Class {
<Do'stona sinflarni e'lon qilish>
private:
<xususiy ma'lumotlar a'zolari>
<xususiy konstruktorlar>
<xususiy usullar>
protected:
<himoyalangan ma'lumotlar a'zolari>
<himoyalangan konstruktorlar>
<himoyalangan usullar>
public:
<ommaviy xususiyatlar>
<ommaviy ma'lumotlar a'zolari>
<ommaviy konstruktorlar>

<ommaviy destruktor>
<ommaviy usullar>

Himoyalangan (**protected**) komponentalar sinf ichida va hosila sinflarda murojaat huquqiga ega.

Sinf o'zining bazaviy sinfidan yuzaga kelayotganida, uning barcha nomlari hosila sinfda avtomatik tarzda yashirin private bo'lib qoladi. Ammo uni, bazaviy sinfning quyidagi kirish spetsifikatorlarini ko'rsatgan holda, osongina o'zgartirish mumkin:

private. Bazaviy sinfning meros bo'lib o'tayotgan (ya'ni himoyalangan va ommaviy) nomlari hosila sinf nusxalarida kirib bo'lmaydigan bo'lib qoladi.

protected. Meros bo'lib o'tayotgan (ya'ni himoyalangan va umumiy) elementlar **protected,** ya'ni himoyalangan murojaat huquqiga ega bo'ladi.

public. Bazaviy sinf va uning ajdodlarining nomlari hosila sinf nusxalarida kirib bo'ladigan bo'ladi, barcha himoyalangan nomlar esa himoyalangan bo'lib qolaveradi.

Bazaviy sinf imkoniyatlarini *kengaytiradigan* sinflarni yuzaga keltirish mumkin: bu yo'l siz uchun g'oyat qulay, ammo ozgina ishlashni talab qilgan funksiyaga ega. Hosila sinfda kerakli funksiyani yangidan yaratish vaqtni bekorga sarflash bilan barobar. Buning o'rniga bazaviy sinfda koddan takroran foydalanish kerak: bunda u talab qilingan darajada kengaytirilishi mumkin. Hosila sinfda sizni qoniqtirmaydigan bazaviy sinf funksiyasini qayta aniqlang xolos.

Xuddi shunday yo'l bilan bazaviy sinf imkoniyatlarini cheklaydigan sinflarni yuzaga keltirish mumkin: Bu yo'l siz uchun g'oyat qulay, ammo nimanidir ortiqcha qiladi.

Quyidagi misolda tavsiflarni kengaytirish va cheklash usullarining qo'llanishini ko'rib chiqamiz:

```
#include <iostream.h>
class Base
{
public:
void display(){cout << "Salom Dunyo. \n";}
};
class Derived: public Base
{
```

```
public:
void display()
{
// asos sinf display() funksiyasini bajaradi
Base::display();
cout <<"Xayr Dunyo. \n";
}
};
int main()
{
Derived d;
d.display(); // hosila sinf display()funksiyasini bajaradi
return 0;
}
```

Natija:

Salom Dunyo.

Xayr Dunyo.

Bu misolning Base:: display() qatorida asos sinf display() usuliga oshkor murojaat qilinadi. Lekin bu yo'l bilan xususiy elementlarga murojaat qilib bo'lmaydi.

Agar asos sinf parametrli konstruktorga ega bo'lsa, hosila sinf ham albatta parametrli kontruktorga ega bo'lishi shart va bu konstruktor initsializatorlar ro'yxatida yoki tanasida asos sinf konstruktorini chaqirishi lozim. Masalan:

```
class Base
{
int a;
public:
Base(int a1): a(a1){}
};
class Derived: public Base
{
int b;
public:
Derived(int a1, int b1): Base(a1), b(b1){}
};
```


Bu misolda hosila sinf konstruktorida asos sinf konstruktori chaqiriladi so'ngra hosila sinf parametrlari initsializatsiya qilinadi.

Parametrlar konstruktor tanasida initsializatsiya qilinishi mumkin.

Masalan:

```
class Base
{
int a;
public:
Base(int a1) {a=a1;};
};
class Derived: public Base
{
int b;
public:
Derived(int a, int b) {Base(a1); b=b1;}
};
```

Obyektlar yaratilganda avval asos sinf konstruktori chaqiriladi, so'ngra hosila sinf konstruktori. Destruktorlar teskari tartibda chaqiriladi.

Masalan:

```
#include <iostream.h>
class Base
{
public:
Base() {cout<<"Base sinfi konstruktori \n";}
~Base() {cout<<"Base sinfi destruktori \n";}
};
class Derived: public Base
{
public:
Derived() {cout<<"Derived sinfi konstruktori \n";}
~Derived() {cout<<"Derived sinfi destruktori \n";}
};
int main ()
{
Derived d;
return 0;
```

```
}
```

Natija:

```
Base sinfi konstruktori
Derived sinfi konstruktori
Derived sinfi destruktori
Base sinfi destruktori
```

Quyida keltirilgan dasturda ikkita oddiy geometrik obyekt – doira va *silindrning sinflar tabaqalanishi* e'lon qilingan.

Dastur shunday tuzilganki, bunda doiraning r – radiusi va silindrning h – balandligi o'zgaruvchilarining ichki qiymatlari yaratilayotgan obyektlar parametrlarini aniqlashi kerak. Circle bazaviy sinfi doirani modellashtiradi, Cylinder hosila sinfi esa silindrni modellashtiradi.

```
#include <iostream.h>
#include <math.h>
const double pi=4 * atan(1);
class Circle {
protected:
double r;
public:
Circle (double rVal=0): r(rVal) {}
void setRadius(double rVal) {r=rVal;};
double getRadius() {return r;};
double Area() {return pi*r*r;};
void showData();
};
class Cylinder: public Circle {
protected:
double h;
public:
Cylinder(double hVal=0, double rVal=0)
: h(hVal), Circle(rVal) {};
void setHeight(double hVal) {h=hVal;}
double getHeight() {return h;};
double Area() {return 2*Circle::Area()+2*pi*r*h;};
void showData();
};
```



```
void Circle::showData()
{
cout <<"Doira radiusi=" <<getRadius()<<endl;
cout<<"Aylana maydoni=" <<Area()<<endl;
};
void Cylinder::showData()
{
cout << "Asos radiusi=" << getRadius()<<endl;
cout<< "TSilindr balandligi=" << getHeight()<< endl;
cout<<"YUza maydoni=" <<Area ()<< endl;
};
int main()
{
Circle circle(2);
Cylinder cylinder(10, 1);
circle.showData ();
cylinder.showData();
return 0;
}
```

Misolda Circle sinfining e'loni r ma'lumotlarining yagona a'zosi, konstruktor va qator usullardan iborat. Obyektni yaratishda konstruktor r ma'lumotlar a'zosini doira radiusining boshlang'ich qiymati bilan nomlaydi (initsiallashtiradi). Konstruktorning yangi sintaksisini ko'rsatib o'tamiz: chaqirishda u bazaviy konstruktor sinfiga, shuningdek, ikki nuqtadan keyin ko'rsatilgan har qanday ma'lumotlar a'zosiga murojaat qilishi mumkin. Bizning misolimizda r ma'lumotlari a'zosi unga rVal parametri bilan murojaat qilish orqali «yaratiladi» va nolli qiymat bilan nomlanadi (initsiallashtiriladi).

Misolda setRadius usuli r ma'lumotlar a'zosi qiymatini belgilaydi, getRadius usuli esa uni qaytaradi. Area usuli doira maydonini qaytaradi. showData usuli aylana radiusi va doira maydonining qiymatlarini chiqarib beradi.

Misolda Circle sinfining hosilasi deb e'lon qilingan Cylinder sinfi h – yagona ma'lumotlar a'zosi, konstruktor va qator usullardan iborat. Bu sinf r ma'lumotlar a'zosini silindr asosi radiusini saqlash uchun hamda setRadius va getRadius usullarini meros qilib oladi. Obyektni yaratishda konstruk-

tor r va h ma'lumotlar a'zolarini boshlang'ich qiymatlar bilan nomlaydi. Konstruktorning yangi sintaksisini ko'rsatib o'tamiz: bizning holatda h ma'lumotlar a'zosi hVal argumentining qiymati bilan nomlanadi (initsiallashtiriladi), r ma'lumotlar a'zosi esa rVal argumentiga ega bo'lgan bazaviy sinf konstruktorini chaqirish bilan nomlanadi.

Misolda setHeight funksiyasi h ma'lumotlar a'zosi qiymatini belgilaydi, getHeight esa qaytaradi. Circle::Area funksiyasi bazaviy sinfdan meros olingan funksiyani, silindr yuzasi maydonini qaytarish uchun ortiqcha yuklaydi. showData funksiyasi esa silindr asosining radiusi, balandligi va yuzining maydoni qiymatlarini chiqarib beradi.

Misolda main funksiyasi Circle sinfiga mansub 2 radiusli Circle aylanasini hamda balandligi 10, asosining radiusi 1 bo'lgan Cylinder sinfiga mansub Cylinder silindrni yaratadi, keyin yaratilgan obyektlarning parametrlarini chiqarish uchun showData ga murojaat qiladi.

Aylana radiusi=2 Doira maydoni=12.566
Asos radiusi=1 silindr balandligi=10 Yuzasining maydoni=69.115

15.2. Ko'plikdagi vorislik

Ko'plik vorislik ta'rifi. Ko'plik vorislik deb sinf ta'rifida bir necha sinf asos sinf sifatida ko'rsatish imkoniga aytiladi.

Masalan:

```
class Basis1
{int a, b;
public:
Basis1(int x, int y){a=x; b=y;}
};
class Basis2
{int c;
char*s;
public:
Basis2(int x, char*y){c=x; b=new char[strlen(y)+1];}
~Basis2(){delete[]s;}
};
class Inherit:public Basis1, public Basis2
{int sum;
public:
```



```
Inherit(int x, int y, int z, char*d, int k):Basis1(x, y), Basis2(z, d)
{sum=k;}
};
```

Quyidagi dasturda computer sinfini yaratish uchun, computer_screen va mother_board asos sinflaridan foydalaniladi:

```
#include <iostream>
#include <string>

using namespace std;
class computer_screen
{
public:
computer_screen(string, long, int, int);
void show_screen(void);
private:
string type;
long colors;
int x_resolution;
int y_resolution;
};
computer_screen::computer_screen(string type, long colors, int x_
res, int y_res)
{
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 << "Ekran turi:" << type << endl;
cout << "Rang:" << colors << endl;
cout << "Kattaligi:" << x_resolution << "ga" << 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 << "Processor:" << processor << endl;
cout << "CHastota:" << speed << "MGC" << endl;
cout << "OZU:" << RAM << "MBayt" << endl;
}
class computer: public computer_screen, public mother_board
{
public:
computer(string, int, float, string, long, int, int, int, int, int);
void show_computer(void);
private:
string name;
int hard_disk;
float floppy;
};
computer::computer(string name, int hard_disk, float floppy,
string 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)
{
computer::name=name;
computer::hard_disk=hard_disk;
```



```
computer::floppy=floppy;
}
void computer::show_computer(void)
{
cout << "Tur:" << name << endl;
cout << "Qattiq disk:" << hard_disk << "MBayt" << endl;
cout << "Yumshoq disk:" << floppy << "MBayt" << endl;
show_mother_board();
show_screen();
}
int main()
{
computer my_pc("Compaq", 212, 1.44, "SVGA",
16000000, 640, 480, 486, 66, 8);
my_pc.show_computer();
return 0;
}
```

Bu misolda computer sinfi konstruktori mother_board i computer_ screen konstruktorlarini chaqiriladi:

computer::computer(string name, int hard_disk, float floppy, string 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)

15.3. Polimorf usullar

Sinflarda polimorfizm. Polimorfizm yuqorida aytilganidek, yunoncha so'z bo'lib, ikkita o'zakdan – poli (ko'p) va morfos (shakl)dan iborat hamda ko'p shakllilikni bildiradi. Polimorfizm – bu turdosh obyektlar (ya'ni bitta ajdod hosilasi bo'lgan sinflarga mansub obyektlar)ning dastur bajarilish vaqtida vaziyatga qarab, o'zlarini turlicha tuta olish xususiyati. OMY doirasida dasturchi obyekt xulq-atvoriga faqat bilvosita ta'sir ko'rsatishi, ya'ni dasturga kiritilayotgan usullarni o'zgartirishi hamda avlodlarga o'z ajdodlarida yo'q bo'lgan o'ziga xos xususiyatlarni qo'shishi mumkin.

Usulni o'zgartirish uchun uni avlodda qo'shimcha yuklash kerak, ya'ni avlodda bitta nomdagi usulni e'lon qilish va unda kerakli xatti-harakatlarni ishga solish kerak. Natijada ajdod-obyekt va avlod-obyektda bitta nomdagi

ikkita usul amal qiladi. Bunda ushbu usullarning kodlari turlicha ishga tushiriladi va, demakki, obyektlar turlicha xatti-harakat ko'rsatadi. Masalan, geometrik shakllar turdosh sinflarining tabaqalanishida (nuqta, to'g'ri chiziq, kvadrat, to'g'ri to'rtburchak, doira, ellips va boshqalar) har bir sinf Draw usuliga ega bo'lib, u ushbu shaklni chizib berish talabi qo'yilgan voqea-hodisaga tegishli javob berilishi uchun mas'uldir.

Polimorfizm tufayli, avlodlar bitta voqeaga o'ziga xos tarzda munosabat bildirish uchun, o'z ajdodlarining umumiy usullarini qo'shimcha yuklashlari mumkin.

Virtual funksiyalar. OMD da polimorfizmga faqat yuqorida tavsifi berilgan vorislik va ajdod usulini qo'shimcha yuklatish mexanizmi vositasida emas, balki *virtuallash* vositasida ham erishiladiki, u ajdod funksiyalarga o'z avlodlari funksiyalariga murojaat qilish imkonini beradi. Polimorfizm sinf arxitekturasi orqali ishga tushiriladi, biroq faqat a'zo-funksiyalar polimorf bo'lishlari mumkin.

C++da polimorf funksiya bitta nomdagi ehtimoliy funksiyalardan biriga faqat bajarilish paytida, ya'ni unga sinfning konkret obyekti uzatilayotgan paytda bog'lab qo'yiladi. Boshqacha qilib aytganda, dastlabki dastur matnida funksiyaning chaqirilishi faqat taxminan belgilanadi, aynan qanday funksiya chaqirilayotgani aniq ko'rsatilmaydi. Bu jarayon kechikkan bog'lanish deb nom olgan. Navbatdagi misol oddiy a'zo-funksiyalarning polimorf bo'lmaganligi nimaga olib kelishi mumkinligini ko'rsatadi.

```
#include <iostream>
#include <string>
using namespace std;
class Parent
{
public:
void F1() {cout<<"I am Parent"<<endl;};
void F2(int n) {
for(int i=0; i<n; i++) F1();
};
};
class Child: public Parent
{
```



```
public:
void F1() {cout<<"I am Parent"<<endl;}
};
int main() {
Child child;
child.F2(3);

int kk;cin>>kk;
return 0;
}
```

Natija:

```
I am Parent
I am Parent
I am Parent
```

Parent sinfi F1 va F2 a'zo-funksiyalarga ega, bunda F2 ni F1 chaqiradi. Parent sinfining hosilasi bo'lgan Child sinfiga F2 funksiyasi vorislikka o'tadi, biroq F1 funksiyasi qayta ta'riflanadi. Kompilator vorislikka o'tgan F2 funksiyani Parent::F1 funksiyasi bilan bog'lab, translatsiya qilib yuboradi.

C++ kechikkan bog'lanishni funksiya bajarilish paytida aniqlaydi hamda funksiyalarni virtuallash vositasida ularda polimorflikni ta'minlaydi. Ajdod va avlod sinflarda virtual funksiyalarni e'lon qilish sintaksisini umumlashtiradigan misolni ko'rib chiqamiz:

```
class className1 {
//Boshqa a'zo-funksiyalar
virtual return Type functionName(<parametrlar ro'yxati>);
}
class className2: public className1 {
//Boshqa a'zo-funksiyalar
virtual return Type functionName(< >);
}
```

Parent va Child sinflari obyektlarida F1 funksiyasi polimorfligini ta'minlash uchun, uni virtual deb e'lon qilish zarur. Quyida dasturning modifikatsiyalangan matni keltiriladi:

```
class Parent
{
public:
virtual void F1() {cout<<"I am Parent" <<endl;};
void F2(int n) {
for(int i=0; i<n; i++) F1();
};
};
class Child: public Parent
{
public:
void F1() {cout<<"I am Parent" <<endl;}
};
int main() {
Child child;
child.F2(3);
return 0;
}
```

Natija:

I am Child
I am Child
I am Child

Mana endi dastur kutilayotgan natijani chiqarib beradi. Kompilator Child.F2(3) ifodasini meros qilib olingan Parent::F2 funksiyasi murojaatiga translatsiya qilib yuboradi, bu funksiya esa, o'z navbatida, Child::F1 avlodining qayta aniqlangan virtual funksiyasini chaqirib oladi.

Agar funksiya bazaviy sinfda virtual deb e'lon qilingan bo'lsa, uni faqat hosila sinflarda qayta aniqlash mumkin, bunda parametrlar ro'yxati avvalgidek qolishi zarur. Agar hosila sinfning virtual funksiyasi parametrlar ro'yxatini o'zgartirgan bo'lsa, bu holda uning bazaviy sinfdagi (hamda uning barcha ajdodlaridagi) versiyasi kirib bo'lmas bo'lib qoladi. Boshida bunday vaziyat boshi berk ko'chaga kirib qolgandek ko'rinishi mumkin. Amalda ortiqcha yuklanish mexanizmini qo'llab-quvvatlamaydigan OMD tillarida shunday bo'ladi ham. C++ bu muammoni virtual funksiyalardan emas, balki xuddi shu nomli,

faqat boshqa parametr ro'yxatiga ega bo'lgan, ortiqcha yuklangan funksiyalardan foydalangan holda hal qiladi.

Virtual deb e'lon qilingan funksiya, hosila sinflarda **virtual** kalit-so'z bilan e'lon qilingani yoki qilinmaganidan qat'iy nazar, barcha hosila sinflarda virtual hisoblanadi.

Virtual funksiyalardan berilgan sinf obyektlarining o'ziga xos xulq-atvorini ishga solish uchun foydalaning. Barcha usullaringizni virtual deb e'lon qilmang, bu ularni chaqirishda qo'shimcha hisoblash sarflariga olib keladi. Hamma vaqt destruktorlarni virtual deb e'lon qiling. Bu sinflar tabaqalanishida obyektlarni yo'q qilishda polimorf xulq-atvorni ta'minlaydi.

Polimorf obyekt-telefonning yaratilishi. Aytaylik, sizning boshliqlaringiz sizga obyekt-telefoningiz diskli, tugmachali yoki to'lovli telefonlardan birini tanlab olib, emulatsiya qila olishi kerak dedi. Boshqacha qilib aytganda, obyekt-telefon bitta qo'ng'iroq uchun tugmachali apparat sifatida, boshqa qo'ng'iroq uchun to'lovli telefon sifatida va h.k. ishlashi mumkin edi. Ya'ni bir qo'ng'iroqdan ikkinchisiga sizning obyekt-telefoningiz o'z shaklini o'zgartirishi lozim bo'ladi.

Turli sinflarga mansub bu telefonlarda faqat bitta farqlanuvchi funksiya mavjud – bu dial usuli. Polimorf obyektni yaratish uchun, siz avval bazaviy sinf funksiyalarini, ularning prototiplari oldidan virtual kalit-so'zni qo'ygan holda, aniqlaysiz. Bu bazaviy sinf funksiyalari hosila sinflar funksiyalaridan shu bilan farqlanadiki, ular virtualdir.

Keyin dasturda parametrli bazaviy sinf obyektiga ilova global funksiya tuziladi. Funksiya tanasida dial usuliga murojaat qilinadi:

```
void dial_phone(phone& this_phone, string this_number)
{this_phone.dial(this_number);};
```

Obyekt shaklini o'zgartirish uchun, siz, quyida ko'rsatilganidek, ushbu funksiyaga hosila sinf obyektini parametr sifatida uzatasiz:

```
dial_phone(home_phone, "303-555-1212");
```

Funksiyada kelgan (phone&) belgisi turlarga keltirishga imkon berib, bir turdagi o'zgaruvchi (touch_tone) adresini boshqa turdagi o'zgaruvchi (phone)ga berish zarurligini ma'lum qiladi. Dastur dial_phone funksiyasiga turli obyektlar adresini taqdim qilishi mumkin ekan, demakki, funk-

siya polimorf bo'lishi mumkin. Navbatdagi dasturda bu usuldan obyekt-telefon yaratish uchun foydalanadi. Dastur ishga tushirilgach, poly_phone obyekti o'z shaklini diskli telefondan tugmachalisiga, keyin esa to'lovlisiga o'zgartiradi:

```
#include <iostream>
#include <string>
using namespace std;
class phone
{
public:
virtual void dial(string number) {cout << "Raqam to'plami" << number << endl;}
void answer(void) {cout << "Javobni kutish" << endl;}
void hangup(void) {cout << "Qo'ng'iroq bajarildi-trubkani qo'yish" << endl;}
void ring(void) {cout << "Qo'ng'iroq, qo'ng'iroq, qo'ng'iroq" << endl;}
phone(string number) {phone::number=number;};
protected:
string number;
};
class touch_tone: public phone
{
public:
void dial(string number) {cout << "Pik Pik Raqam to'plami" << number << endl;}
touch_tone(string number): phone(number) {}
};
class pay_phone: public phone
{
public:
void dial(string number) {cout << "Iltimos to'lang" << amount << "sent" << endl;
cout << "Raqam to'plami" << number << endl;}
pay_phone(string number, int amount): phone(number) {pay_phone::amount=amount;}
private:
```



```
int amount;
};

void dial_phone(phone& this_phone, string this_number)
{this_phone.dial(this_number);};

int main()
{
pay_phone city_phone("702-555-1212", 25);
touch_tone home_phone("555-1212");
phone rotary("201-555-1212");
// Obyekt diskli telefonga aylantirilsin
dial_phone(rotary, "818-555-1212");
// Obyekt shakli tugmachali telefonga o'zgartirilsin
dial_phone(home_phone, "303-555-1212");
// Obyekt shakli to'lovli telefonga o'zgartirilsin
dial_phone(city_phone, "212-555-1212");
return 0;
}
```

*Agar ushbu dastur kompilatsiya qilinib ishga tushirilsa, displey ekrani-*da quyidagi yozuv paydo bo'ladi:

```
S:\> POLYMORP <ENTER>
Raqam to'plami 818-555-1212
Pik Pik Raqam to'plami 303-555-1212
Iltimos to'lang 25 sent
Raqam to'plami 212-555-1212
```

poly_phone obyekti dastur bajarilishi davomida o'z shaklini o'zgartirib turar ekan, u polimorf bo'ladi.

15.4. Abstrakt sinflar

Abstrakt sinf ta'rifi. Hech bo'lmaganda bitta sof (bo'sh) virtual *funksiyaga ega bo'lgan sinf abstrakt sinf deyiladi.*

Quyidagi e'longa ega bo'lgan komponentali funksiya sof virtual funksiya deyiladi:

virtual <tur> <funksiya_nomi>

```
(<formal_parametrlar_ro'yxati>)=0;
```

Bu yozuvda "=0" konstruksiya "sof spesifikator" deyiladi. Sof virtual funksiya ta'rifiga misol:

```
virtual void fpure (void)=0;
```

Sof virtual funksiya hech narsa qilmaydi va uni chaqirib bo'lmaydi. Uning qo'llanilishi – hosila sinflarda uning o'rnini egallovchi funksiyalar uchun asos bo'lish. Abstrakt sinf esa hosila sinf uchun asosiy (bazaviy) sinf sifatida ishlatilishi mumkin. Agar sof virtual funksiya hosila sinfda to'liq ta'riflanmasa, u hosila sinfda ham sof virtual bo'lib qoladi, natijada hosila sinf ham abstrakt sinf bo'ladi.

Abstrakt sinfni faqat boshqa sinf ajdodi sifatida ishlatish mumkin:

Ba'zi sinflar abstrakt tushunchalarni ifodalaydi va ular uchun obyekt yaratib bo'lmaydi. Bunday sinflar biror hosila sinfda ma'noga ega bo'ladi.

Masalan,

```
class Abstract
{
public:
virtual void draw()=0;
};
class Derived: public Abstract
{
public:
void draw() {cout <<"Salom.";}
};
int main(void)
{
Derived d;
Abstract a;
return 0;
}
```

Agar sof virtual funksiya hosila sinfda to'liq ta'riflanmasa, u hosila sinfda ham sof virtual bo'lib qoladi, natijada hosila sinf ham abstrakt sinf bo'ladi.

```
class Abstract
{
virtual int f()=0;
```



```
virtual float g(float)=0;
};
class Derived: class Abstract
{
int f();
};
int main (void)
{
Abstract a; //xato
Derived d; // xato
}
```

Abstrakt sinflar realizatsiya detallarini aniqlashtirmasdan faqat interfeysni ko'rsatish uchun ishlatiladi. Masalan, operatsion tizimda qurilma drayveri abstrakt sinf sifatida berilishi mumkin:

```
class character_device {
public:
virtual int open()=0;
virtual int close(const char*)=0;
virtual int read(const char*, int)=0;
virtual int write(const char*, int)=0;
virtual int ioctl(int ...)=0;
// ...
};
```

Drayverlar character_device sinfining ajdodlari sifatida kiritilishi mumkin.

Abstrakt sinf turidagi o'zgaruvchi yaratib bo'lmaydi, lekin abstrakt sinf turidagi ko'rsatkich yaratish mumkin. Bu ko'rsatkichga abstrakt sinf abstrakt bo'lmagan turli avlodlari adresini qiymat sifatida berib, abstrakt usulga mos turli usullarni chaqirish mumkin.

Masalan, yuqoridagi polimorf telefon quyidagicha ta'riflanishi mumkin:

```
#include <iostream>
#include <string>
using namespace std;
class abstract_phone
```

```
{
public:
virtual void dial(string number)=0;
void answer(void) {cout << "Javobni kutish" << endl;}
void hangup(void) {cout << "Qo'ng'iroq bajarildi-trubkani qo'yish"
<< endl;
}
void ring(void) {cout << "Qo'ng'iroq, qo'ng'iroq, qo'ng'iroq" <<
endl;}
abstract_phone(string number) {abstract_phone::number=num-
ber;};
protected:
string number;
};
class phone:public abstract_phone
{
public:
virtual void dial(string number) {cout << "Raqam to'plami" <<
number << endl;}
phone(string number): abstract_phone(number) {};
};
class touch_tone: public abstract_phone
{
public:
void dial(string number) {cout << "Pik Pik Raqam to'plami" <<
number << endl;}
touch_tone(string number): abstract_phone(number) {}
};
class pay_phone: public abstract_phone
{
public:
void dial(string number) {cout << "Iltimos to'lang" << amount <<
"sent" << endl;
cout << "Raqam to'plami" << number << endl;}
pay_phone(string number, int amount): abstract_phone(number)
{pay_phone::amount=amount;}
```



```
private:
int amount;
};
void dial_phone(abstract_phone& this_phone, string this_
number)
{this_phone.dial(this_number);};
int main()
{
pay_phone city_phone("702-555-1212", 25);
touch_tone home_phone("555-1212");
phone rotary("201-555-1212");
// Obyekt diskli telefonga aylantirilsin
dial_phone(rotary, "818-555-1212");
// Obyekt shakli tugmachali telefonga o'zgartirilsin
dial_phone(home_phone, "303-555-1212");
// Obyekt shakli to'lovli telefonga o'zgartirilsin
dial_phone(city_phone, "212-555-1212");
return 0;
}
```

15-bob bo'yicha savollar

1. Vorislik nima uchun kerak?
2. Himoyalangan protected va xususiy private huquqlari orasida qanday farq bor?
3. Nima uchun avval ajdod sinf konstruktorlari chaqirilib, so'ngra avlod sinf konstruktori chaqiriladi?
4. Nima uchun destruktorlar konstruktorlarga nisbatan teskari tartibda chaqiriladi?
5. Vorislikda ajdod sinf spetsifikatori sifatida protected ko'rsatilishi mumkinmi?
6. Sinflar kutubxonasini qurishda vorislikdan qanday foydalaniladi?
7. Xususiy deb e'lon qilingan komponentalarga boshqa sinf usullari orqali murojaat qilish mumkinmi?
8. Polimorfizm deb nimaga aytiladi?
9. Usulni virtual deb e'lon qilish qanday imkon yaratadi?
10. Abstrakt sinf deb qanday sinfga aytiladi?

standart tur, sinf
sinf, standart tur
sinf, sinf

Masalan: Complex sinfi uchun quyidagi dostona operatsiya-funksiyalarni kiritish mumkin:

```
complex operator + (Complex x, Complex y)
{
return (Complex(x.real +y.real, x.imag +y.imag());
}

Complex operator+ (double x, Complex y)
{
return (Complex(x+y.real, y.imag());
}

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

Quyidagi dasturda ifodalar qo'llanilgan:

```
#include <iostream>
using namespace std;
class Complex
{
double re, im;
public:
Complex(){};
Complex(double re1, double im1)
{
re=re1; im=im1;
}
void show()
{
cout<<"re="<<re<<"im="<<im<<endl;
};
friend Complex operator+ (Complex a, Complex b);
```



```
friend Complex operator+ (double x, Complex b);
friend Complex operator+(Complex a, double y);
};
Complex operator+ (Complex a, Complex b)
{
return (Complex(a.re+b.re, a.im+b.im));
};
Complex operator+ (double x, Complex b)
{
return (Complex(x+b.re, b.im));
}
Complex operator+(Complex a, double y)
{
return (Complex (a.re+y, a.im));
};

int main ()
{
Complex CC (1.0, 2.0);
Complex EE(3.0,4.0);
Complex DD=CC+EE;
DD.show();
EE=EE+2.0;
EE.show();
return 0;
}
```

Natija:

re=4 im=6
re=5 im=4

Standart turlarni sinf obyektiga keltirish vazifasini konstruktorga topshirish mumkin. Masalan, Complex sinfiga quyidagi konstruktorni qo'yish hamma yordamchi funksiyalardan xalos bo'lish imkonini beradi:

```
Complex (double x)
{
real=x; imag=0.0;
}
```

bo'la olmaydi. Misol uchun AA – biror sinf obyekti 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) xatoga olib keladi.

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

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

Quyidagi dasturda prefiks ++ va -- hamda postfiks ++ va -- operatsiyalarini qo'shimcha yuklash ko'rsatilgan:

```
#include <iostream>
using namespace std;
class Pair
{
int N;
double x;
friend Pair& operator ++(Pair&);
friend Pair& operator ++(Pair&, int);
public:
Pair (int n, double xn)
{
N=n; x=xn;
}
void display ()
{
cout<<"\n Koordinatalar: N="<<N<<"\tx="<<x;
}
Pair& operator --()
{
N/=10; x/=10;
return Pair(N, x);
};
```



```
Pair& operator --(int k)
{
N/=2;
x/=2.0;
return Pair (N, x);
}
};
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;
}
int main ()
{
Pair Z(10, 20.0);
Z.display();
++Z;
Z.display();
--Z;
Z.display();
Z++;
Z.display();
Z--;
Z.display();
return 0;
}
```

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 ++ esa 2 marta oshirishni bildiradi. Prefiks -- qiymatni 10 marta kamaytirish, postfiks -- esa qiymatni 20 marta kamaytirishni bildiradi.

Turlarni o'zgartirish operatori. Yuqorida konstruktor yordamida standart turdagi, o'zgaruvchi haqiqiy o'zgaruvchi sinf turidagi, ya'ni kompleks turidagi o'zgaruvchiga keltirishni ko'rdik. Teskarisini amalga oshirish uchun turni o'zgartirish operatoridan foydalanish mumkin. Quyidagi misolda kasr son sinfi kiritilib, kasr sonni haqiqiy songa keltirish operatori kiritilgan:

```
#include <iostream>
using namespace std;
class Fraction
{
int s,m;
public:
Fraction(int s1, int m1=1){s=s1; m=m1;};
void show(){cout<<s<<'/'<<m<<endl;};
operator double()
{
return (double)s/m;
}
};
int main ()
{
Fraction D(1,2);
D.show();
double c=D;
cout<<c<<endl;
D=1;
D.show();
return 0;
}
```

Natija:

1/2
0.5
1/1

16.2. Shablonlar

Funksiya shabloni. Shablonlar (parametrlangan turlar) bog'langan funksiyalar yoki sinflar oilasini tuzish imkonini beradi. Shablonni aniqlashning umumiy sintaksisi quyidagi ko'rinishga ega:

template <shablonli turlar ro'yxati> {e'lon qilish};

Funksiyalar shablonlari va *sinflar shablonlari* farqlanadi.

Funksiya shabloni qo'shimcha yuklanayotgan funksiyalarni aniqlash namunasini beradi. Solishtirishga yo'l qo'yadigan har qanday turdagi ikkita argumentdan kattarog'ini qaytaradigan max (x, y) funksiyasi shablonini ko'rib chiqamiz:

template <class T> T max(Tx, Ty){**return**(x>y)? x:y;};

bunda <class T> shablonining argumenti tomonidan taqdim etilgan ma'lumotlar turi har qanday bo'lishi mumkin. Undan dasturda foydalanishda kompilator max funksiyasi kodini bu funksiyaga uzatilayotgan parametrlarning faktik turiga muvofiq generatsiya qiladi.

Misol:

```
#include <iostream>
using namespace std;
template <class T>
T maximum(T x, T y)
{
if (x>y) return x; else return y;
};
int main()
{
int i=3;
float a=3.0, b=7.5;
i=maximum(i, 0);//argumentlar turi int
cout<<i<<endl;
float m=maximum(a, b);// argumentlar turi float
cout<<m<<endl;
return 0;
}
```

Natija:

3
7.5

Faktik turlar kompilatsiya paytida ma'lum bo'lishlari kerak. Shablonlarsiz max funksiyasini ko'p martalab qo'shimcha yuklashga to'g'ri kelar edi, ya'ni, garchi barcha funksiya versiyalarining kodlari bir xil bo'lsa ham, har bir qo'llanilayotgan tur uchun alohida ortiqcha yuklash kerak bo'lar edi. C++ standarti bu maqsad uchun **#define max(x, y) ((x>y)? x:y)** makrosidan foydalanishni qat'iyan tavsiya etmaydi, chunki C++ tiliga oddiy C tili ustidan shunday afzalliklarni beradigan turlarni tekshirish mexanizmi blokirovka qilingan bo'lishi mumkin. Shu narsa ayonki, max(x, u) funksiyasining vazifasi – bir-biriga mos turlarni qiyoslash. Afsuski, makrosni qo'llash bir-biriga mos kelmaydigan turlarning (masalan, int va struct) qiyoslanishiga yo'l qo'yadi.

```
//ikkita o'zgaruvchining o'rnini o'zgartiradigan funksiya shabloni:
template<class T> //T – parametrlanayotgan tur nomi
void change(T&x, T&y)
{
T z=x;
x=y;
y=z;
}
```

Bu funksiya quyidagicha chaqirilishi mumkin:

```
long k=10, i=5;
change(k, i);
```

Kompilator:

void change(long& x, long& y){long z=x; x=y; y=z;} ta'rifini ifodalab beradi.

Funksiyalar shablonlari parametrlarining asosiy xususiyatlari:

1. Parametrlar nomlari shablonning butun ta'rifi bo'ylab unikal bo'lmog'i lozim.
2. Shablon parametrlarining ro'yxati bo'sh bo'la olmaydi.
3. Shablon parametrlari ro'yxatida har biri class so'zidan boshlanadigan bir nechta parametr bo'lishi mumkin.

Funksiya shablonini qo'shimcha yuklash. Funksiya shablonini qo'shimcha yuklash mumkin. Masalan, ikki butun son qo'shilganda, albatta, haqiqiy son hosil bo'lishi kerak bo'lsin:

```
template<class T> T add (T a, T b)
```



```
{
return a+b;
}
float add (int a, int b)
{
return (float) a+b;
}
```

Endi add funksiyasini ikki butun son uchun chaqirilsa ikkinchi funksiyaga murojaat qilinadi, qolgan hollarda funksiya shabloniga murojaat qilinadi.

Yuqoridagi funksiya shablonlarida bitta umumiy turdan foydalanilgan. Ko'p hollarda funksiya shablonida bir nechta tur ko'rsatish talab etiladi. Quyidagi misolda show_array funksiya shablonidan foydalanilgan bo'lib, massiv elementlarini chiqarish uchun mo'ljallangan. T tur massiv turini aniqlash, T1 tur count parametri turini ko'rsatish uchun ishlatiladi. Dasturda int va float turidagi massivlar chiqariladi.

```
#include <iostream>
using namespace std;
template<class T,class T1>
void show_array(T array[],T1 count)
{
T1 index;
for (index=0; index<count; index++) cout << array[index] << " ";
cout << endl;
}
int main()
{
int pages[]={100, 200, 300, 400, 500};
float prices[]={10.05, 20.10, 30.15};
show_array(pages, 5);
show_array(prices, 3);
return 0;
}
```

Natija:

100, 200, 300, 400, 500

10.05, 20.10, 30.15

Sinflar shabloni. Sinflar shabloni sinflar oilasini aniqlash namunasini beradi. Quyida bir o'lchamli ma'lumotlar massivi sinflarining generatori bo'lgan massiv shabloniga misol keltirilgan:

template <class T>class array

Ushbu sinfning turlashtirilgan elementlari ustida, elementlarning konkret turidan qat'iy nazar, bir xil bazaviy operatsiyalar (kiritilsin, o'chirilsin, indekslansin va h.k.) bajariladi. Agar turga parametrdek muomala qilinsa, bu holda kompilator berilgan tur elementlariga ega bo'lgan vektorlar sinflarini generatsiya qiladi.

Navbatdagi dastur ikkita sinfni yaratishda array sinf shablonidan foydalanadi. Bu sinflarning biri int turining qiymatlari bilan, ikkinchisi float turining qiymatlari bilan ish ko'radi.

```
#include <iostream>
#include <fstream>
using namespace std;
const int MAX=1000;
template<class T, class T1>
class Array
{
public:
Array(int max_size);
int add_value(T);
T sum();
T1 average_value();
void show_array();
private:
T data[MAX];
int max_size;
int index;
};
template<class T, class T1>
Array<T, T1>::Array(int size)
{
```



```
if (size>=MAX)
{
cerr << "Xotira yetarli emas – dastur tugayapti" << endl;
exit(1);
}
Array::max_size=size;
Array::index=0;
};

template<class T, class T1>
int Array<T, T1>::add_value(T value)
{
if (index== max_size)
return(-1); // Massiv to'la
else
{
data[index]=value;
index++;
return(0); // Omadli
}
}
template<class T, class T1>
T Array<T, T1>::sum()
{
T sum=0;
for (int i=0; i<index; i++) sum +=data[i];
return(sum);
}
template<class T, class T1>
T1 Array<T, T1>::average_value()
{
T1 sum=0;
for (int i=0; i<index; i++) sum +=data[i];
return (sum / index);
}
template<class T, class T1>
void Array<T, T1>::show_array()
```