2-Ma'ruza.

2-Mavzu. Qurilish va arxitektura soxasida axborot jarayonlarini algoritmlash va dasturlash.

Reja:

1. C++ dasturlash tilining yaratilishi haqida ma`lumot.
2. C++ tilining xizmachi so`zlari.
3. C++ tilida o`zgarmaslar va o`zgaruvchilar.

C++ dasturlash tili *C* dasturlash tiliga asoslangan. *C* dasturlsh tili o`z navbatida *B* va *BCPL* dasturlashgan tillaridan kelib chiqqan. *BCPL* - 1967 yilda *Martin Richards* tomonidan o`ylab topilgan bo`lib, operatsion tizimlarni yaratish uchun mo`ljallangan. *Ken Thompson* o`zining *B* tilida *BCPL* ning ko`p xossalarini yaratishga harakat qilgan va *B* dasturlash tilida asosan operatsion tizimning birinchi variantlarini yozgan. *BCPL* ham, *B* ham tipsiz til bo`lgan. Yani o`garuvchilarning ma`lum bir tipi bo`lmagan - har bir o`zgaruvchi kompyuter xotirasida faqat bir bayt joy egallagan. O`zgaruvchini qanday sifatda ishlatish esa, yani butun sonmi, haqiqiy sonmi yoki harfmi, dasturchining vazifasi bo`lgan.

C tilini *Dennis Ritchie B* tiliga asoslanib yaratdi va ilk bor *C* tilini 1972 yili Bell Laboratoriyasida, *DEC PDP-11* kompyuterida qo`lladi. *C* o`zidan oldingi *B* va *BCPL* tillarining juda ko`p muhim tomonlarini o`z ichiga olish bilan bir qatorda o`zgaruvchilarni tiplashtiradi va turli yangiliklar kiritilgan. Boshlanishda *C* asosan *UNIX* tizimlarida keng tarqaldi. C mashina arxitekturasi bilan tez muloqot qiluvchi dasturlash tilidir. 1983 yilda, *C* tili keng tarqalganligi sababli, uni standartlash harakati boshlandi. Buning uchun Amerika Milliy Standartlar Komiteti (*ANSI*) qoshida *X3J11* texnik komitet tuzildi. 1989 yilda ushbu standart qabul qilindi. Standartni dunyo bo`yicha keng tarqatish maqsadida 1990 yilda *ANSI* va Dunyo Standartlar Tashkiloti (ISO) hamkorlikda *C* ning ANSI/ISO 9899:1990 standartini qabul qilishdi. Shuning uchun *C* da yozilgan dasturlar mayda o`zgarishlar yoki umuman o`zgarishlarsiz juda ko`p kompyuter platformalarida ishlaydi.

C++ 1980 -yillar boshida *Bjarne Stroustrup* tomonidan *C* ga asoslangan tarzda tuzildi. *C++* juda ko`p imkoniyatlarni o`z ichiga olgan, lekin eng asosiysi u ob`yektlar asosida dasturlashga imkon beradi. Dasturlarni tez va sifatli yozishga hozirgi kunda katta ahamiyat berilmoqda. Buni ta`minlash uchun ob`yektli dasturlash g`oyasi ilgari surildi. Xuddi 1970 - yillar boshida strukturali dasturlash kabi, dasturlarni hayotdagi jismlarni modellashtiruvchi ob`yektlar orqali tuzish dasturlash sohasida inqilob qildi. *C++* dan tashqari boshqa ko`p ob`yektli dasturlashga yo`naltirilgan tillar mavjud. *C++* esa *gibrid* tildir. Unda *C* ga o`xshab strukturali dasturlash yoki yangicha, ob`yektlar bilan dasturlash mumkin. Yangicha deyishimiz ham nisbiydir. Ob`yektli dasturlash falsafasi paydo bo`lganiga ham yigirma yildan oshyapti. *C++* funksiya va ob`yektlarning juda katta kutubxonasiga ega. Yani *C++* tilida dasturlashni o`rganish ikki qismga bo`linadi. Birinchisi bu *C++* ni o`zini o`rganish, ikkinchisi esa *C++* ning standart kutubxonasidagi tayyor ob`ekt-funksiyalardan foydalanishni o`rganishdir.

***C++* alfaviti.** *C++* tilida buyruqlar va so`zlar, barcha elementlar *C++* tilining alfavitida yoziladi. Alfavitga quyidagi simvollar kiradi.

- Katta va kichik lotin alfaviti harflari **(*A, B, … Z, a, b, … z*),** pastga chiziqcha belgisi (_) (harflar bilan barovar yozilganda);
- arab raqamlar: **0** dan **9** gacha;
- Maxsus simvollar; masalan, **+, *, { , &;**
- Ko`rinmaydigan simvollar (“Umumlashgan bo`shliq simvollari”). Leksemalarni o`zaro ajratish uchun ishlatiladigan simvollar (masalan, bo`shliq, tabulyatsiya, yangi qatorga o`tish belgilari).

C++ alfaviti aslida kompyuterdagi barcha belgilarni qabul qiladi. Chunki standartda uning alfavitini barcha belgilar to`plami deb qabul qilingan. Shuning uchun izohlarda, satrlarda va simvolli o`zgarmaslarda boshqa literallar, masalan, rus harflarini ishlatilishi mumkin. *C++*

tilida olti xil turdagi leksemalar ishlatiladi: erkin tanlanadigan va ishlatiladigan identifikatorlar, xizmatchi so`zlar, o`zgarmaslar (const), amallar, ajratuvchi belgilar.

Identifikator. Identifikator bu – dastur ob`yektining nomi. Identifikatorlar lotin harflari, ostki chiziq belgisi va sonlar ketma - ketligidan iborat bo`ladi. Identifikator lotin harfidan yoki ostki chizish belgisidan boshlanishi lozim. Masalan, *a, b, _t, _A*. Identifikatorlarning uzunligi standart bo`yicha chegaralanmagan. Katta va kichik harflar farqlanadi, shuning uchun oxirgi ikki identifikator bir biridan farq qiladi. *Borland* kompilyatorlaridan foydalanilganda nomning birinchi 32 harfi , ba`zi kompilyatorlarda 8 ta harfi inobatga olinadi. Bu holda NUMBER_OF_TEST va NUMBER_OF_ROOM identifikatorlari bir biridan farq qilmaydi. Identifikatorlar tilining maxsus (xizmatchi) so`zlari bilan mos bo`lmasligi lozim. Identifikatorlarni past chiziq bilan e`lon qilish maslahat berilmaydi.

**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 xizmachi so`zlar mavjud:

***1-Jadval. C++* tilining xizmachi so`zlari.**

asm	do	if	return	typedef
auto	double	inline	short	typeid
bool	dynamic_ cast	int	signed	typename
break	else	long	sizeof	union
case	enum	mutable	static	unsigned
catch	explicit	namespace	static_cast	Using
char	export	new	Struct	virtual
class	extern	operator	switch	void
const	false	private	template	volatile
const_cast	float	protected	this	wchar_t
continue	for	public	throw	While
default	friend	register	true	

Amallar. Amallar bir yoki bir nechta belgilar bilan aniqlanadi va operatorlar ustida bajariladi. Amal orasida bo`sh joy qo`yilmaydi. Amaldagi belgilar maxsus belgilardan (masalan, &&, |, <) va harflardan (masalan, reinterpret_cast, new) iborat bo`lishi mumkin.

Operandlar soniga qarab amallar uch guruhga bo`linadi: **UNARY, BINARY, TERNARY**. Standart amallar qayta aniqlanadi.

**O`zgarmaslar.** *C++* tilida o`zgarmaslar o`zgarmas kattalikdir. Ularning mantiqiy, butun, haqiqiy, belgili, satrli o`zgarmaslarga bo`linadi. Dasturchi *C++* tilida o`zgarmaslarni aniq ifodalay olishi kerak.

2-Jadval. O`zgarmaslar formati.

O`zgarmas	O`zgarmas formati	misol
mantiqiy	True va false so`zlar bilan aniqlanadi	True, False
butun	O`nlik sanoq sistemasi. Birinchi raqami 0 bo`lishi kerak emas (0,1,2,3,4,5,6,7,8,9)	15, 25, 0, 4
	Sakkizlik sanoq sistemasi. Birinchi raqami 0 bo`lishi kerak (0,1,2,3,4,5,6,7)	01, 020, 07155
	O`n oltilik sanoq sistamasi. Boshlanishi 0x (0X) bilan bo`lishi kerak (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E)	0xA, 0x1B8, 0X00FF, 0X00ff
Haqiqiy	O`nli. [son].[son] ko`rinishda	5.7, .001, 35
	Exponensial. [son][.][son]{E|e}[+|-][son]	0.2E6, .11e-3, 5E10, 1.22E-10
Belgili	Tirnoq [`] ichiga olingan bir yoki bir nechta belgi	`A`, `yu`, `*`, `db`, `A`, `\n`, `\012`, `\x07\x07`

Satrli	Belgilarning qo`shtirnoqqa olingani	"Salom Buxoro", "\tNatija =\xF5\n"

**Satrli o`zgarmas**. Satrli o`zgarmaslar orasiga eskeyp simvollarni qo`llash mumkin. Bu simvollar oldiga [\] belgisi quyiladi. Masalan, “\n Birinchi satr \n ikkinchi satr \n uchinchi satr”. Satr simvollari xotirada ketma-ket joylashtiriladi va har bir satrli o`zgarmas oxiriga avtomatik ravishda kompilyator tomonidan `\0` simvoli qo`shiladi. Shunday satrning xotiradagi hajmi simvollar soni+1 baytga tengdir. Ketma-ket kelgan va bo`shliq, tabulyatsiya yoki satr oxiri belgisi bilan ajratilgan satrlar kompilyatsiya davrida bitta satrga aylantiriladi. Masalan, “Salom” “Buxoro ” satrlari bitta satr deb qaraladi. “Salom Buxoro”. Bu qoidaga bir necha qatorga yozilgan satrlar ham bo`ysinadi.

Sanovchi o`zgarmas. C++ tilining qo`shimcha imkoniyatlaridan biri.

Sanovchi o`zgarmaslar ENUM xizmatchi so`zi yordamida kiritilib, butun tipdagi sonlarga qulay so`zlarni mos qo`yish uchun ishlatiladi. Masalan,

```
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 o`zgarmaslar quyidagi qiymatlarni qabul qiladi:

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

O`zgarmaslar aralash ko`rinishda kiritilishi ham mumkin:

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

Bu misolda avtomatik ravishda o`zgarmaslar quyidagi qiymatlarni qabul qiladi:

```
Zero=0, one=1, for=4;five=5,seeks=6;
Enum BOOLEAN {NO, YES};
```

O`zgarmaslar qiymatlari: NO=0, YES=1.

Nomlangan o`zgarmaslar. C++ tilida o`zgaruvchilardan tashqari nomlangan o`zgarmaslar kiritilishi mumkin. Bu o`zgarmaslar qiymatlarini dasturda o`zgartirish mumkin emas. O`zgarmaslar 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 o`zgarmaslar quyidagi shaklda kiritiladi:

```
Const tip o`zgarmas_nomi=o`zgarmas_qiymati
```

Masalan,

```
Const double Pi=3.1415;
Const long M=99999999;
Const R=2;
```

Oxirgi misolda o`zgarmas tipi ko`rsatilmagan, bu o`zgarmas int tipiga tegishli deb hisoblanadi.

Null ko`rsatkich. NULL - ko`rsatkich yagona arifmetik bo`lmagan o`zgarmasdir. Null ko`rsatkich 0 yoki 0L yoki nomlangan o`zgarmas NULL orqali tasvirlanishi mumkin. Shuni aytish lozimki bu o`zgarmas qiymati 0 bo`lishi yoki `0` simvoli kodiga mos kelishi shart emas.

O`zgarmaslar chegaralari va mos tiplari.

O`zgarmas turi	Ma`lumotlar tipi	Hajm, bayt	Qiymatlar chegarasi
mantiqiy	bool	1	True, false
belgili	signed char	1	-128…127
	Unsigned char	1	0…255
Sanovchi	Enum	2	-32768…32767
butun	signed short int	2	-32 768 … 32 767
	unsigned short int	2	0…65535

O`zgarmas turi	Ma`lumotlar tipi	Hajm, bayt	Qiymatlar chegarasi
mantiqiy	bool	1	True, false
	signed int	4	-2 147 483 648 … 2 147 483 647
	Unsigned int	4	0 … 4 294 967 295
	signed long int	4	-2 147 483 648 … 2 147 483 647
	unsigned long int	4	0 … 4 294 967
haqiqiy	Float	4	3.4E-32…3.4E+38
	Double	8	1.7E-308…1.7E+308
	Long double	10	3.4E-4932…1.1E+4932

**O`zgaruvchilar (VARIABLES).** O`zgaruvchilar ob`yekt sifatida qaraladi. *C++* tilining asosiy tushunchalaridan biri nomlangan xotira qismi – ob`yekt tushunchasidir. Ob`yektning xususiy holi bu o`zgaruvchidir. O`zgaruvchiga qiymat berilganda unga ajratilgan xotira qismiga shu qiymat kodi yoziladi. O`zgaruvchi qiymatiga nomi orqali murojat qilish mumkin, xotira qismiga esa faqat manzili orqali murojat qilinadi. O`zgaruvchi nomi bu erkin kiritiladigan identifikatordir. O`zgaruvchi nomi sifatida xizmatchi so`zlarni ishlatish mumkin emas.

O`zgaruvchilar tiplari.

Bool	Mantiqiy
Char	bitta simvol
long char	uzun simvol
short int	qisqa butun son
Int	butun son
long int	uzun butun son
float	haqiqiy son
double (long float)	ikkilangan haqiqiy son
long double	uzun ikkilangan haqiqiy son

Butun sonlar ta`riflanganda ko`rilgan tiplar oldiga *unsigned* (ishorasiz) ta`rifi ko`rinishida bo`lishi mumkin. Bu ta`rif qo`shilgan butun sonlar ustida amallar *mod 2n* arifmetikasiga asoslangandir. Bu erda *n* soni int tipi xotirada egallovchi razryadlar sonidir. Agar ishorasiz *k* soni uzunligi int soni razryadlar sonidan uzun bo`lsa, bu son qiymati *k mod 2n* ga teng bo`ladi. Ishorasiz *k* son uchun ga *–k* amali *2n – k* formula asosida hisoblanadi. Ishorali ya`ni *signed* tipidagi sonlarning eng katta razryadi son ishorasini ko`rsatish uchun ishlatilsa *unsigned* (ishorasiz) tipdagi sonlarda bu razryad sonni tasvirlash uchun ishlatiladi. O`zgaruvchilarni dasturning ixtiyoriy qismida ta`riflash yoki qayta ta`riflash mumkin. Masalan,

```
Short int a; Short int b1; Short int ac;
int a; int b1; int ac;
```

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

```
int I=0;
char c=`k`;
```

Typedef ta`riflovchisi yangi tiplarni kiritishga imkon beradi. Masalan, yangi KOD tipini kiritish:

```
typedef unsigned char KOD;
KOD simbol;
```

C++ tilida amallar sakkiz guruhga bo`linadi. Ular quyidagi jadvalda keltirilgan.

Amallar.

Arifmetik amallar	Razryadli amallar	Nisbat amallari	Mantiqiy amallar
[+] qo`shish	[&] va	[==] teng	[&&] va
[-] ayrish	[|] yoki	[!=] teng emas	[||] yoki

[*] ko`paytirish	[^] inkor	[>] katta	[!] inkor
[/] bo`lish	[<<] chapga surish	[>=] katta yoki teng	
[%] modul olish			
[-] unar minus			
[+] unar plyus	[>>] o`ngga surish	[<] kichik	
[++] birga oshirish		[<=] kichik yoki teng	
[--] birga kamaytirish	[~] inkor		

Imlo amallar	Qiymat berish va shartli amallar	Tipli amallar	Manzilli amallar
[()] – doirali qavs	[=] - oddiy qiymar berish	[(tip)] – tipni o`zgartirish	[&] - manzilni aniqlash
[[]] – kvadrat qavs	[op=] - murakkab qiymat berish	sizeof- hajmni hisoblash	[*] - manzil bo`yicha qiymat aniqlash yoki joylash
[,] - vergul	[?] – shartli amal		

C++ da arifmetik amallar. Ko`p dasturlar bajarilishi davomida arifmetik amallarni bajaradi.

Arifmetik amal.

Arifmetik amal	Arifmetik operator	Algebraik ifoda	C++ dagi ifodasi
Qo`shish	+	A+B	A+B
Ayirish	-	A-B	A-B
Ko`paytirish	*	AB	A*B
Bo`lish	/	A/B	A/B
Modul olish	%	A MOD B	A % B

**Ba`zi bir xususiyatlar**. Butun sonli bo`lishda, yani bo`luvchi ham, bo`linuvchi ham butun son bo`lganda, javob butun son bo`ladi. Javob yaxlitlanmaydi, kasr qismi tashlanib yuborilib, butun qismining o`zi qoladi.

Modul operatori [%] butun songa bo`lishdan kelib shiqadigan qoldiqni beradi. *x%y* ifodasi *x* ni *y* ga bo`lgandan keyin chiqadigan qoldiqni beradi. [%] operatori faqat butun sonlar bilan ishlaydi. Haqiqiy sonlar bilan ishlash uchun "*math.h*" kutubxonasidagi *fmod* funksiyasini qo`llash kerak.

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

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

Amallar odatda **unary** ya`ni bitta operandga qo`llaniladigan amallarga va **binary** ya`ni ikki operandga qo`llaniladigan amallarga ajratiladi.

Binar amallar additiv ya`ni qo`shuv [+] va ayirish [–] amallariga , hamda multiplikativ ya`ni ko`paytirish [*], bo`lish [/] va modul olish[%] amallariga ajratiladi. Additiv amallarining ustuvorligi multiplikativ amallarining ustuvorligidan pastroqdir. Butun sonni butun songa bo`lganda natija butun songacha yaxlitlanadi. Masalan, 10/3=3, (-10)/3=-3, 10/(-3) =-3.

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 kompilyatorga bog`liqdir.

Binar arifmetik amallar bajarilganda tiplarni keltirish quyidagi qoidalar asosida amalga oshiriladi:

- *short* va *char* tiplari int tipiga keltiriladi;

- agar operandlar biri *long* tipiga tegishli bo`lsa ikkinchi operand ham *long* tipiga keltiriladi va natija ham *long* tipiga tegishli bo`ladi;
- agar operandlar biri *float* tipiga tegishli bo`lsa ikkinchi operand ham *float* tipiga keltiriladi va natija ham *float* tipiga tegishli bo`ladi;
- agar operandlar biri *double* tipiga tegishli bo`lsa ikkinchi operand ham *double* tipiga keltiriladi va natija ham *double* tipiga tegishli bo`ladi;
- agar operandlar biri *long double* tipiga tegishli bo`lsa ikkinchi operand ham *long double* tipiga keltiriladi va natija ham *long double* tipiga tegishli bo`ladi;

Unar amallarga ishorani o`zgartiruvchi *unar minus* [–] va *unar plus* [+] amallari kiradi. Bundan tashqari [++] va [--] amallari ham unar amallarga kiradi.

[++] *unar* amali qiymatni 1 ga oshirishni ko`rsatadi. Amalni *prefiks* ya`ni *++i* ko`rinishda ishlatish oldin o`zgaruvchi qiyMatni oshirib so`ngra foydalanish lozimligini, postfiks ya`ni *i++* ko`rinishda ishlatishdan oldin o`zgaruvchi qiymatidan foydalanib, so`ngra oshirish kerakligini ko`rsatadi. Masalan, *i* qiymati 2 ga teng bo`lsin, u holda *3+(++i)* ifoda qiymati 6 ga, *3+i++* ifoda qiymati 5 ga teng bo`ladi. Ikkala holda ham i qiymati 3 ga teng bo`ladi.

[--] unar amali qiymatni 1 ga kamaytirishni ko`rsatadi. Bu amal ham prefiks va postfiks ko`rinishda ishlatilishi mumkin. Masalan, i qiymati 2 ga teng bo`lsin, u holda --i ifoda qiymati 1 ga, i-- ifoda qiymati 2 ga teng bo`ladi. Ikkala holda ham i qiymati 1 ga teng bo`ladi.

Bu ikki amalni faqat o`zgaruvchilarga qo`llash mumkindir. Unar amallarning ustivorligi *binar* amallardan yuqoridir.

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

6&5 qiyjmati 4 ga ya`ni 100 ga teng.
6|5 qiyjmati 7 ga ya`ni 111 ga teng.
6^5 qiymati 3 ga ya`ni 011 ga teng.
~6 qiyjmati 2 ga ya`ni 010 ga teng.

Bu misollarda amallar ustivorligi oshib borishi tartibida berilgandir.

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, , yani 5<<0=5 , 5<<1 =10, 5<<2=20 mos keladi.

Agar operand musbat bo`lsa N pozitsiyaga o`ngga surish chap operandni ikkining N chi 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 qolgan amallar binar amallarga kiradi.

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

1!=0 qiymati 1 ga teng;
1==0 qiymati 0 ga teng;
3>=3 qiymati 1 ga teng;
3>3 qiymati 0 ga teng;
2<=2 qiymati 1 ga teng;
2<2 qiymati 0 ga teng;

Katta [>], kichik [<], katta yoki teng [>=], kichik yoki teng [<=] amallarining ustivorligi bir xildir.

Teng [==]va teng emas [!=] amallarining ustivorligi o`zaro teng va qolgan amallardan pastdir.

Mantiqiy amallar. Mantiqiy amallar asosan butun sonlarga qo`llanadi. 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 berilgandir. Inkor [!] amali unar qolganlari binar amallardir.

Qiymat berish amali. Qiymat berish amali [=] binar amal bo`lib chap operandni odatda o`zgaruvchi o`ng operandi odatda ifodaga teng bo`ladi. Masalan, Z=4.7+3.34;

Bitta ifodada bir necha qiymat berish amallari qo`llanilishi mumkin. Masalan, C=y=f=4.2+2.8;.

Bundan tashqari C ++ tili da murakkab qiymat berish amali mavjud bo`lib, umumiy ko`rinishi quyidagichadir:

O`zgaruvchi_nomi amal= ifoda;

Bu erda amal quyidagi amallardan biri bo`lishi mumkin: *,/,%,+,-, &,^,|, <<,>>. Masalan,

X+=4 ifoda x=x+4 ifodaga teng kuchlidir;

X*=a ifoda x=x*a ifodaga teng kuchlidir;

X/=a+b ifoda x=x/(a+b) ifodaga teng kuchlidir;

X>>=4 ifoda x=x>>4 ifodaga teng kuchlidir;

Imlo belgilari amal sifatida. C ++ tilida ba`zi bir imlo belgilari ham amal sifatida ishlatilishi mumkin. Bu belgilardan oddiy () va kvadrat [] qavslardir. Oddiy qavslar binar amal deb qaralib ifodalarda yoki funksiyaga murojat qilishda foydalaniladi. Funksiyaga murojat qilish quyidagi shaklda amalga oshiriladi: <funksiya nomi> (<argumentlar ro`yxati>). Masalan, sin(x) yoki max(a,b).

Kvadrat qavslardan massivlarga murojat qilishda foydalaniladi. Bu murojat quyidagicha amalga oshiriladi: <massiv nomi>[<indeks>]. Masalan, a[5] yoki b[n][m].

Vergul simvolini ajratuvchi belgi deb ham qarash mumkin amal sifatida ham qarash mumkin. Vergul bilan ajratilgan amallar ketma-ketligi bir amal deb qaralib, chapdan o`ngga hisoblanadi va ohirgi ifoda qiymati natija deb qaraladi. Masalan, d=4,d+2 amali natijasi 8 ga teng.

Shartli amal. Shartli amal ternar amal deyiladi va uchta operanddan iborat bo`ladi: <1-ifoda>?<2-ifoda>:<3-ifoda>. Shartli amal bajarilganda avval 1- ifoda hisoblanadi. Agar 1-ifoda qiymati 0 dan farqli bo`lsa 2- ifoda hisoblanadi va qiymati natija sifatida qabul qilinadi, aks holda 3-ifoda hisoblanadi va qiymati natija sifatida qabul qilinadi. Masalan, modulni hisoblash: x<0?-x:x yoki ikkita son kichigini hisoblash a<b?a:b .

Shuni aytish lozimki shartli ifodadan har qanday ifoda sifatida foydalanish mumkin. Agar F FLOAT tipga, N – INT tipga tegishli bo`lsa, (N > 0) ? F : N

ifoda N musbat yoki manfiyligidan qat`iy nazar DOUBLE tipiga tegishli bo`ladi. Shartli ifodada birinchi ifodani qavsga olish shart emas.

Tiplar bilan ishlovchi amallar. Tiplarni o`zgartirish amali quyidagi ko`rinishga ega: (tip_nomi) operand; Bu amal operandlar qiyMatni ko`rsatilgan tipga keltirish uchun ishlatiladi. Operand sifatida o`zgarmas, o`zgaruvchi yoki qavslarga olinga ifoda kelishi mumkin. Misol uchun (long)6 amali o`zgarmas qiyMatni o`zgartirmagan holda operativ xotirada egallagan baytlar sonini oshiradi. Bu misolda o`zgarmas tipi o`zgarmagan bo`lsa, (double) 6 yoki (float) 6 amali o`zgarmas ichki ko`rinishini ham o`zgartiradi. Katta butun sonlar haqiqiy tipga keltirilganda sonning aniqligi yo`qolishi mumkin.

sizeof amali operand sifatida ko`rsatilgan ob`yektning baytlarda xotiradagi hajmini hisoblash uchun ishlatiladi. Bu amalning ikki ko`rinishi mavjud: sizeof ifoda sizeof (tip) Misol uchun:

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

Amallar ustivorligi

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

Dastur tuzilishi. Dastur komandalar 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:

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

Main funksiyasi ikki usulda ishlatilishi mumkin (tipli va tipsiz). Yuqorida keltirilgan misolda tipsiz edi. Tipli main ga dastur quyidagi ko`rinishda tuziladi:

```
Preprocessor_komandalari
 tip main()
{ Dastur tanasi.
return [qiymat] }
```

Preprocessor direktivalari kompilyatsiya jarayonidan oldin preprocessor tomonidan bajariladi. Natijada dastur matni preprocessor direktivalari asosida o`zgartiriladi. Preprocessor komandalaridan ikkitasini ko`rib chiqamiz. *# include <fayl_nomi>* Bu direktiva standart kutubxonadagi funksiyalarni dasturga joylash uchun foydalaniladi. *#define <almashtiruvchi ifoda> <almashinuvchi ifoda>* Bu direktiva bajarilganda dastur matnidagi almashtiruvchi ifodalar almashinuvchi ifodalarga almashtiriladi. Masalan,

1-listing.	Output:
#include <iostream.h> void main()	 Salom, BUXORO!

```
{
  cout << "\n Salom, BUXORO! \n";
}
```

Define direktivasi yordamida bu dasturni quyidagicha yozish mumkin:

2-listing.	Output:
#include <iostream.h> #define program cout << "\n Salom, BUXORO! \n" #define bosh { #define tam } void main() bosh program; tam	Salom, BUXORO!

Define direktivasidan nomlangan o`zgarmaslar kiritish uchun foydalanish mumkindir. Masalan,

```
#define max 10
```

Agar dasturda quyidagi amallar mavjud bo`lsin:

```
Double m=max
A=alfa*max
```

Preprocessor bu matnda har bir max o`zgarmasni uning qiymati bilan almashtiradi, va natijada quyidagi amallar hosil bo`ladi.

```
Double max=10
D=alfa*10
```

Dastur matni va preprocessor. C++ tilida matnli fayl shaklida tayyorlangan dastur uchta qayta ishlash bosqichlaridan o`tadi. Matnni preprocessor direktivalari asosida o`zgartilishi. Bu jarayon natijasi yana matnli fayl bo`lib preprocessor tomonidan bajariladi.

Kompilyatsiya. Bu jarayon natijasi mashina kodiga o`tkazilgan ob`yektli fayl bo`lib, kompilyator tomonidan bajariladi.

**Bog`lash.** Bu jarayon natijasi to`la mashina kodiga o`tkazilgan bajariluvchi fayl bo`lib, bog`lagich tomonidan bajariladi.

Preprocessor vazifasi dastur matnini preprocessor direktivalari asosida o`zgartirishdir. *Define* direktivasi dasturda bir jumlani ikkinchi jumla bilan almashtirish uchun ishlatiladi. Bu direktivadan foydalanishning sodda misollarini biz yuqorida ko`rib chiqdik. *Include* direktivasi ikki ko`rinishda ishlatilishi mumkin. *#include* fayl nomi direktivasi dasturning shu direktiva o`rniga qaysi matnli fayllarni qo`shish kerakligini ko`rsatadi. *#include <fayl nomi>* direktivasi dasturga kompilyator standart kutubxonalariga mos keluvchi sarlavhali fayllar matnlarini qo`shish uchun mo`ljallangandir. Bu fayllarda funksiya prototipi, tiplar, o`zgaruvchilar, o`zgarmaslar ta`riflari yozilgan bo`ladi. Funksiya prototipi funksiya qaytaruvchi tip, funksiya nomi va funksiyaga uzatiluvchi tiplardan iborat bo`ladi. Masalan, *cos* funksiyasi prototipi quyidagicha yozilishi mumkin: double *cos(double).* Agar funksiya nomidan oldin *void* tipi 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, kompilyatsiya bosqichidan so`ng amalga oshiriladi.

Kompilyatsiya bosqichida sintaksis hatolar tekshiriladi va dasturda bunday hatolar 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 lozimdir. Shuning uchun bu fayllarga sarlavhali fayl (*header file*) nomi berilgandir.

Dasturda kiritish va chiqarish funksiyalaridan masalan, *cout*<< funksiyasidan foydalanish uchun *#include <iostream.h>* direktivasidan foydalanish lozimdir. Bu direktivada

iostream.h sarlavhali fayl nomi quyidagilarni bildiradi: st- standart, i- input(kirish), o- output(chiqish), h – head(sarlavha).

Mantiqiy solishtirish operatorlari. C++ bir necha solishtirish operatorlariga ega.

Mantiqiy solishtirish operatorlari.

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

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

Yuqoridagi solishtirish operatorlarini ishlatadigan bir misolni ko`raylik.

3-listing.

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

Output:

```
Ikki sonni kiriting: 10
5
10 katta yoki teng 5 ga
10 teng emas 5 ga
```

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

if (shart) ifoda;

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

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

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

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

Har qanday dastur funksiyalar ketma ketligidan iborat bo`ladi. Funksiyalar sarlavha va funksiya tanasidan iborat bo`ladi. Funksiya sarlavhasiga *void main()* ifoda misol bo`la oladi. Funksiya tanasi ob`yektlar ta`riflari va operatorlardan iborat bo`ladi.

Har qanday operator nuqta-vergul [;] belgisi bilan tugashi lozim. Quyidagi ifodalar *X=0*, yoki *I++* operatorga aylanadi agar ulardan so`ng nuqtali vergul [;] kelsa (*X = 0; I++;)*.

Operatorlar bajariluvchi va bajarilmaydigan operatorlarga ajratiladi. Bajarilmaydigan operator bu izoh operatoridir. Izoh operatori [/*] belgisi bilan boshlanib, [*/] belgisi bilan tugaydi. Bu ikki simvol orasida ixtiyoriy jumla yozish mumkin. Kompilyator 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. Ma`lumotlarni o`zgartiruvchi operatorlarga qiymat berish operatorlari va [;] bilan tugovchi ifodalar kiradi. Masalan,

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

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

- Qo`shma operatorlar;
- Tanlash operatorlari;
- Takrorlash operatorlari;
- O`tish operatorlari;

**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 ob`yektlar ta`riflari mavjud bo`lishi mumkin. Quyidagi dastur qismi qo`shma operator:

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

Bu fragment bo`lsa blok:

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

Kiritish-chiqarish operatorlari. Chiquvchi oqim *cout* kelishilgan bo`yicha ekranga mos keladi. Lekin maxsus operatorlar yordamida oqimni printer yoki faylga mos qo`yish mumkin.

4-listing.	Output:

#include <iostream.h>
void main(void)
{
  cout << 1001;
}
	1001

4a-listing.	**Output:**
#include <iostream.h> void main(void) (cout << 1 << 0 << 0 << 1; }	1001

Kiruvchi oqim *cin* kelishilgan bo`yicha ekranga mos keladi.

5-listing.	Output:

#include <iostream.h> void main(void) { int a cin >> a; cout << a*a; }	a*a
5a-listing.	Output:
#include <iostream.h> void main(void) (int a,b; cin >> a >> b; cout << a*b; }	a*b

Qiymat berish operatorlari. Bu qismda keyingi bo`limlarda kerak bo`ladigan tushunchalarni berib o`tamiz. *C++* da hisoblashni va undan keyin javobni o`zgaruvchiga beruvchi bir necha operator mavjuddir. Masalan,

k = k * 4; ni k *= 4;

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

Birga oshirish va kamaytirish operatorlari (INCREMENT and DECREMENT). *C++* da bir argument oluvchi inkrenet (++) va dekrement (--) operatorlari mavjuddir. Bular ikki ko`rinishda ishlatiladi, biri o`zgaruvchidan oldin (++f - preinkrement, --d - predekrement), boshqasi o`zgaruvchidan keyin (s++ - postinkrement, s-- - postdekrement) ishlatilgan holi.

Postinkrementda o`zgaruvchining qiymati ushbu o`zgaruvchi qatnashgan ifodada shlatiladi va undan keyin qiymati birga oshiriladi. Preinkrementda esa o`zgaruvchining qiymati birga oshiriladi, va bu yangi qiymat ifodada qo`llaniladi. Predekrement va postdekrement ham aynan shunday ishlaydi Lekin qiymat birga kamaytiriladi. Bu operatorlar faqatgina o`zgaruvchining qiyMatni birga oshirish, kamaytirish uchun ham ishlatilinishi mumkin, yani boshqa ifoda ichida qo`llanilmasdan. Bu holda pre va post formalarining farqi yo`q. Masalan,

++r; r++;

Yuqoridagilarning funksional jihatdan hech qanday farqi yo`q, chunki bu ikki operator faqat r ning qiyMatni oshirish uchun qo`llanilmoqda. Bu operatorlarni oddiy holda yozsak:

r = r + 1; d = d - 1;

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

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

Dasturdagi o`zgaruvchilar e`lon qilindi va boshqang`ich qiymatlarni olishdi. *cout << k++ << endl;* ifodasida ekranga oldin *k* ning boshlang`ich qiymati chiqarildi, keyin esa uning qiymati 1 da oshirildi. *l += 4;* da l ning qiymatiga 4 soni qo`shildi va yangi qiymat l da saqlandi. cout << *--m* << *endl;* ifodasida m ning qiymati oldin predekrement qilindi,va undan so`ng ekranga chiqarildi. m = k + (++l); da oldin l ning qiymati birga ishirildi va l ning yangi qiymati k ga qo`shildi. m esa bu yangi qiymatni oldi.

```
++(f * 5);
```

ko`rinish noto`g`ridir.

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

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

Bu yerda {} qavslardagi ifodalar bloki faqat i 10 dan kichkina va l 20 dan katta yoki teng bo`lgandagina ijro qilinadi.

AND (&&).

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

Boshqa misol:

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

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

10- jadval. OR (||).

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

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

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

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

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

NOT (!).

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

Nazorat savollari

4. C++ dasturlash tilining yaratilishi haqida ma`lumot.
5. C++ tilining xizmachi so`zlari.
6. C++ tilida o`zgarmaslar va o`zgaruvchilar