Alfavit C#

C# dasturlash tili alfaviti ASCII kodlar tablitsasi simvollarida iborat. C# alfavitiga quyidagilar kiradi:

- lotin alifbosining katta va kichik harflari
- 0 dan 9 gacha bo`lgan raqamlar
- “_” simvoli
- " { } , | [ ] + - % / \ ; ' : ? < > = ! & # ~ . ^ * maxsus simvollar
- boshqa simvollar

Barcha sonlar, lotin tilining katta va kichik harflari va “_” simvoli C# alfavitining harflari hisoblanadi.

C# alfaviti leksemelarni hosil qilish uchun xizmat qiladi. Leksmelar bu dasturlash tilida ishlatiladigan so`zlar hisoblanadi. 5 turdagi leksemalar mavjud:

- identifikatorlar(идентификаторы);
- kalit so`zlar(ключевые слово);
- operatsiya belgilari (simvollar)(знаки (символы) операций);
- literallar(литералы);
- Ajratuvchi simvollar(разделители);

Barcha leksema turlari (kalit so`zlar va identifikatorlardan tashqari) o`zining so`z hosil qilish qoidasiga va o`zining maxsus alfavitiga ya`ni C# alfaviti to`plamning qismiga ega.

Leksemelar ajratuvchi simvollar oraqali bir-biridan ajratiladi. Bu simvollar ko`pgina probel(bo`sh joy)lar ketma-ketligi, bitta probel, yangi satr simvoli, tabulyatsiya(табуляция) va izohlar(комментария) hisobladi.

Identifikatorlarni hosil qilish qoidasi

Identifikatorlarni alfavit harflaridan hosil qilish qoidasini qarab chiqamiz.

- identifikatorning birinchi simvoli harf yoki “_” simvoli bo`lishi kerak;
- keyingi simvollar harf, raqam va “_” simvolidan iborat bo`lishi mumkin;
- identifikator uzunligi(simvollar soni) chegaralanmagan;

C# da “_” simvolini identifikatorning birinchi simvoli sifatida ishlatish mumkin.

Obyektlani nomlash bo`yicha maslahatlar.

Nomlar - bu identifikatorlar. Har qanday harf, raqam va "_" simvollaridan tuzilgan va birinchi simvoli harf bo`lgan ketma-ketlik til gramatikasi bo`yicha har qanday obyektning nomi bo`la oladi. Lekin har qanday simvollar ketma-ketligidan hosil qilingan nom ma`nosiz va dastur kodini tushunishga qiyin holatga olib kelishi mumkin. Shuning uchun obyekt nomi sifatida shu obyekt haqida biror ma`noni anglatuvchu simvollar ketma-ketligidan foydalanish maqsadga muvofiq.

Kalit so`zlar va nomlar

C# tili identifikatorlarining bir qismi kalit so`zlar lug`ati tarkibiga kiradi. Bu identifikatorlar kalit so`zlarning to`plam ostini tashkil qiladi. Identifikatorlar e`lon qilingandan keyin ular nom hisoblanadi. Bu nomlar o`zgaruvchilarni, funksiyalarni, ma`lumotlar tipini nomlash uchun ishlatiladi. Quyida kalit so`zlar jadvali keltirilgan. Bu so`zlarni sinflar, funksiyalar, o`zgaruvchilar va boshqa til strukturalaini hosil qilish yoki nomlashda ishlatilmaydi.

abstract	do	in	protected	true
as	double	int	public	try
base	else	interface	readonly	typeof
bool	enum	internal	ref	uint
break	event	is	return	ulong
byte	explicit	lock	sbyte	unchecked
case	extern	long	sealed	unsafe
catch	false	namespace	short	ushort
char	finally	new	sizeof	using
checked	fixed	null	stackalloc	virtual
class	float	object	static	void
const	for	operator	string	volatile
continue	foreach	out	struct	while
decimal	goto	override	switch	
default	if	params	this	
delegate	implicit	private	throw	

Izohlar

Ba`zan dastur kodida shunday tekstlarni yozishga to`g`ri keladiki, bu tekstlar dastur kodini o`quvchi uchun mo`ljallangan izoh bo`lib, u kompilyator tomonidan inobatga olinmaydi. C# da buni ikki yul bilan bajarish mumkin:

/* simvollari izohni boshlaydi. */ simvollari izohni tugatadi. Bu ko`p qatorli izohlarni hosil qilishda, dasturning bir qismini tahrir qilishda foydalanish mumkin. /* */ izohlar ichma-ich bo`lmaydi.

// simvollari izohni boshlaydi. Izoh o`zi boshlangan qator oxirida tugaydi. Bu usul qisqa izohlarni hosil qilishda foydali.

Literallar

C# da quyidagi turdagi literallar mavjud:

- butun sonli(целочисленный) литерал;(integer-literal)
- haqiqiy sonli(вещественный) литерал; (real-literal)
- simvolli(символьный) литерал; (character-literal)
- satrli(строковый) литерал; (string-literal)
- mantiqiy(логический) literal;(boolean-literal)
- “bo`sh” literal;(null-literal)

Literallar tilning o`zgacha so`z kategoriyasi hisoblanadi. Har bir literallar to`plam osti uchun o`zining so`z hosil qilish qoidasi mavjud. Har bir literal turi strukturasining umumiy tasnifini ko`rib chiqamiz.

Butun sonli literallar butun sonli qiymatlarni yozish uchun xizmat qiladi. Bunday literallar raqamlar ketma-ketligidan iborat. “-” belgisi ham ishlatilishi mumkin. Agar butun sonli literal 0x yoki 0X bilan boshlansa, bu 16 lik sanoq sistemasidagi butun son deb qabul qilinadi. Bunda butun sonli literal A yoki a dan boshlab F yoki f gacha bo`lgan simvollardan o`z ichiga olishi mumkin. 16 lik sanoq sistemasidagi bu simvollar mos ravishda 10 lik sanoq sistemasidagi 10 dan 15 gacha sonlar bilan ekvivalent. Literaldan keyin bevosita U u L l UL Ul uL ul LU Lu lU lu butun son tipi suffikslari joylashgan bo`lishi mumkin.

Butun sonli literallar tipi quyidagicha aniqlanadi:

- agar butun sonli literal suffiksga ega bo`lmasa, unda literalning tipi uning qiymatini o`z ichiga olgan quyidagi tiplardan birinchisi hisoblanadi: int, uint, long, ulong. Misol: 2147483 literali int tipiga, 2247483647 uint tipiga tegishli bo`ladi.
- agar literal U yoki u suffiksiga ega bo`lsa, unda literalning tipi uning qiymatini o`z ichiga olgan quyidagi tiplardan birinchisi hisoblanadi: uint, ulong;
- agar literal L yoki l suffiksiga ega bo`lsa, unda literalning tipi uning qiymatini o`z ichiga olgan quyidagi tiplardan birinchisi hisoblanadi: long, ulong;
- agar literal UL, Ul, uL, ul, LU, Lu, lU, yoki lu suffiksiga ega bo`lsa, unda literalning tipi ulong hisoblanadi;

“l” simvoli o`rnida “L” simvolidan foydalanish maqsadga muvofiq, chunki “l” harfi bilan “1” raqamini chalkashtirib yuborishning oldini oladi.

Haqiqiy sonli literal haqiqiy sonlarni ifodalash uchun xizmat qiladi.

Bu literal *float, double* va *decimal* tipi qiymatlarini yozish uchun ishlatiladi. Literal quyidagi ko`rinishlarda bo`ladi.

- mos ravishda raqamlar ketma-ketligi nuqta yana raqamlar ketma-ketligi, eksponenta qismi va haqiqiy son tipi suffiksidan iborat;
- mos ravishda nuqta, raqamlar ketma-ketligi, eksponenta qismi va haqiqiy son tipi suffiksidan iborat;
- mos ravishda raqamlar ketma-ketligi, eksponenta qismi va haqiqiy son tipi suffiksidan iborat;
- mos ravishda raqamlar ketma-ketligi va haqiqiy son tipi suffiksidan iborat;

Eksponenta qismi mos ravishda e yoki E, + yoki -, raqamlar ketma-ketligidan iborat. M: e-13, E10.

Haqiqiy son tipi suffikslari: F f D d M m.

Haqiqiy sonli literal tipi quyidagicha aniqlanadi:

- agar literal suffiksga ega bo`lmasa, literal *double* tipiga tegishli.
- agar literal F yoki f suffiksiga ega bo`lsa, literal *float* tipiga tegishli.M: 1f, 1.5f, 1e10f, 123.456F
- agar literal D yoki d suffiksiga ega bo`lsa, literal *double* tipiga tegishli.M: 1d, 1.5d, 1e10d, 123.456D

- agar literal M yoki m suffiksiga ega bo`lsa, literal *decimal* tipiga tegishli.M: 1m, 1.5m, 1e10m, 123.456M

Simvolli literal faqatgina bitta simvolni ifodalash uchun ishlatiladi va har doim ikki tomonida bittalik qo`shtirnoq bo`ladi: 'simvol'.

Simvol sifatida quyidagilar bo`lishi mumkin:

- ' (U+0027), \ (U+005C) va yangi satr simvolidan tashqari har qanday simvol;
- oddiy o`tish ketma-ketligi(simply-escape-sequence). U quyidagilardan biri: \' \" \\ \0 \a \b \f \n \r \t \v;
- 16 lik sanoq sistemasidagi o`tish ketma-ketligi(*hexadecimal-escape-sequence*). Uning ko`rinishi: \x 16 lik s.s. dagi raqam, 16 lik s.s. dagi raqam,16 lik s.s. dagi raqam, 16 lik s.s. dagi raqam. Teskari slesh(\) simvolidan keyin kelgan simvol quyidagilardan biri bo`lishi mumkin: ', ", \, 0, a, b, f, n, r, t, u, U, x, v. Boshqa holda kompilyator xatolik haqida xabar beradi. 16 lik sanoq sistemasidagi o`tish ketma-ketligi bitta Unicode simvolini ifodalaydi va uning qiymati (\x) dan keyin yoziladi. Agar simvolli literal qiymati U+FFFF dan oshib ketsa, kompilyator xatolik ko`rsatadi;
- Unicode simvoli simvolli literalda U+0000 dan U+FFFF gacha oraliqda bo`lishi kerak.

Oddiy o`tish ketmaketligi simvolining Unicode simvolidagi ko`rinishi quyidagi jadvalda keltirilgan:

Escape sequence	Character name	Unicode encoding
`\'`	Bittalik qo`shtirnoq	`0x0027`
`\"`	Ikkitalik qo`shtirnoq	`0x0022`
`\\`	Orqa slesh	`0x005C`
`\0`	Null (bo`sh qiymat)	`0x0000`
`\a`	Alert	`0x0007`
`\b`	Backspace	`0x0008`
`\f`	Form feed	`0x000C`
`\n`	Yangi qator	`0x000A`
`\r`	Carriage return	`0x000D`
`\t`	Gorizontal tabulyasiya	`0x0009`
`\v`	Vertikal tabulyasiya	`0x000B`

Bitta simvolni turli ko`rinishda ifodalash mumkin: M: 'Z'='\x5A'. Simvolli literal tipi char hisoblanadi.

C# da ***satrli literallar*** ikki shaklga ega: doimiy satrli literallar(regular string literal) va so`zma-so`z satrli literal(verbatim strig literal).

Doimiy satrli literal bo`sh yoki bir necha simvollardan (" (U+0022), \ (U+005C) va yangi satr simvollaridan tashqari) iborat bo`ladi va ikkitalik qo`shtirnoq bilan chegaralangan bo`ladi. U yana oddiy o`tish ketma-ketligi(tab simvoli \t kabilarni) simvolini, 16 lik sanoq sistemasidagi va Unicode o`tish ketma-ketligi simvolini o`z ichiga olishi mumkin.

So`zma-so`z satrli literallar ikkitalik qo`shtirnoq undan keyin mos ravishda @ simvoli, bo`sh yoki simvollar ketma-ketligi, yopuvchi ikkitalik qo`shtirnoqdan iborat. Bunda oddiy o`tish ketma-ketligi simvoli, 16 lik sanoq sistemasi va Unicode o`tish ketma-ketligi simvoli ishlatilmaydi. So`zma-so`z satrli literallar bir necha qatorda joylashishi mumkin. Teskari slesh(\) simvolidan keyin kelgan simvol quyidagilardan biri bo`lishi mumkin: ', ", \, 0, a, b, f, n, r, t, u, U, x, v. Boshqa holda kompilyator xatolik haqida xabar beradi. Turli ko`rinishdagi satrli literallarga misol:

```
string a = "hello, world";                 // hello, world
string b = @"hello, world";                // hello, world
string c = "hello \t world";               // hello     world
string d = @"hello \t world";              // hello \t world
string e = "Joe said \"Hello\" to me";     // Joe said "Hello" to me
string f = @"Joe said ""Hello"" to me";    // Joe said "Hello" to me
string g = "\\\\server\\share\\file.txt";  // \\server\share\file.txt
string h = @"\\server\share\file.txt";     // \\server\share\file.txt
string i = "one\r\ntwo\r\nthree";
string j = @"one
two
three";.
```

Sartrli literal "\x123" 16 lik sanoq sistemasidagi 123 ga mos keladigan simvolni ifodalaydi. 16 lik sanoq sistemasidagi 12 ga mos simvol va uning ketidan kelgan 3 simvolini o`z ichiga olgan satrni hosil qilish uchun "\x00123" yoki "\x12" + "3" yozish kerak.

Satrli literal tipi string hisoblanadi.

Mantiqiy literal qiymati 2 ta: true va false;

Mantiqiy literal tipi bool.

null-literal:
null
null literal tipi null tipi.

C# da ma`lumotlar tipi

C# tiplashgan til hisoblanadi. Har bir obyektni hosilqilishda uning tipini e`lon qilish zarur (masalan butun son, haqiqiy son, forma, oyna, tugma(knopka), satr va boshqalar). Shu orqali kompilyator xatolardan xoli bo`lishni, ya`ni o`zgaruvchi qabul qilishi mumkin bo`lgan qiymatlarni qabul qilishini ta`minlaydi. Obyekt tipi kompilyatorga obyekt o`lchamini (masalan int tipidagi obyekt xotiradan 4 bayt egallaydi), uning xususiyatlarini (masaslan forma ko`rinadigan va ko`rinmaydigan bo`lishi mumkin) ko`rsatadi.

Xuddi C++ va Java tillari kabi C# da ham tiplar 2 guruhga ajratiladi: Oldindan aniqlangan tilning ichki tiplari va foydalanuvchi (dasturchi) tomonidan aniqlanadigan tiplar.

C# yana tiplarni 2 kategoriyaga bo`ladi: o`lchovli tiplar va ko`rsatqichli tiplar. Ular orasidagi asosiy farq qiymatlarini xotirada saqlash usuli. O`lchvli tiplar qiymatlarini stekda saqlanadi. Ko`rsatqichli tiplar obyektning faqat adresini stekda, o`zini esa kuchada saqlaydi. Kucha dasturning asosiy xotirasi hisolanadi. Kuchaga murojaat qilish stekka murojaat qilishdan ko`ra sekin. Katta obyektlarni kuchada saqlash ko`pgini imkoniyatlarga ega. Bu haqda keyinchalik to`xtalamiz.

Stek ma`lumotlar strukturasi. U elementarni “birinchi kelgan, oxirgi ketadi” prinsipi bo`yicha saqlaydi. Stek prosessor tomonidan qo`llab quvvatlanadigan xotira sohasiga tegishli. Unda lokal o`zgaruvchilar saqlanadi. Stekka murojaat umumiy xotira sohasiga murojaatga nisbatan bir necha marotaba tezroq. Shuning uchun ma`lumotlarni stekda saqlash dasturning ishlash tezligini oshiradi. C# da o`lchovli tiplar(masalan butun sonlar) qiymatini stekda saqlaydi va unga murojaat o`zgaruvchi nomi bilan amalga oshiriladi.

Ko`rsatqichli tiplar kuchada joylahadi. Kucha kompyuterning tezkor xotirasi. Unga murojaat stekka murojaatga nisbatan sekinroq. Agar obyekt kuchada joylashgan bo`lsa, o`zgaruvchi uning faqat adresini saqlaydi. Bu

adres stekda saqlanadi. Adres orqali dastur obyektga murojaat qiladi. Stekda joylashgan har bir o`zgaruvchi ko`rinish sohasidan chiqqan zahoti “сборщик мусора” tomonidan o`chiriladi. Funksiya tanasida e`lon qilingan lokal o`zgaruvchilar runksiya ishini tugatishi bilan o`chiriladi. Kuchada joylashgan obyektlar ham “сборщик мусора” tomonidan o`chiriladi. Bu obyektni ko`rsatib turgan barcha ko`rsatqichlar bilan obyekt orasidagi aloqa uzilganda amalga oshiriladi.

Avvaldan mavjud ichki tiplar

C# tili CLS(Common Language Specification) ga mos keladigan keng tiplar to`plamini taqdim etadi. Bu tiplar NET platformasiga mos keladi yani C# da hosil qilingan obyektlar NET CLS qo`llab quvvatlaydigan har qanday dasturlash tilida foydalanish mumkin (masalan VB.NET). Har bir tip aniq o`zining o`zgarmas o`lchamiga ega. Quyida C# tomonidan taqdim etilgan tiplar ro`yxati jadvalda keltirilgan:

Tip	Qiymatlar sohasi	O`lchovi
sbyte	-128 dan 127 gacha	ishorali butun 8 bit
byte	0 dan 255 gacha	ishorasiz butun 8 bit
char	U + 0000 dan U + ffff gaha	16-bitli Unicode simvoli
bool	true или false.	1 bayt
short	-32,768 dan 32,767 gacha	Ishorali butun 16 bit
ushort	0 dan 65,535 gacha	Ishorasiz butun 16 bit
int	-2,147,483,648 dan 2,147,483,647 gacha	Ishorali butun 32 bit
uint	0 dan 4,294,967,295 gacha	Ishorasiz butun 32 bit
long	-9,223,372,036,854,775,808 dan 9,223,372,036,854,775,807 gacha	Ishorali butun 32 bit
ulong	0 dan 18,446,744,073,709,551,615 gacha	ishorasiz butun 32 bit
float	$\pm 1.5 \times 10^{-45}$ dan $\pm 3.4 \times 10^{38}$ gacha	4 bayt, aniqlik — 7 разряд
double	$\pm 5.0 \times 10^{-324}$ dan $\pm 1.7 \times 10^{308}$ gacha	8 bayt, aniqlik — 16 разряд
decimal	$\pm 1.0 \times 10^{-28}$ dan $\pm 7.9 \times 10^{28}$ gacha	12 bayt, aniqlik — 28 разряд

Yuqoridagiga qo`shimcha tarzda C# da yana enum, struct tipidagi obyektlar ham bor.

Avvaldan mavjud bo`lgan tiplarni o`girish

Bir tip obyekti ikkinchi tip obyektiga bilvosita yoki bevosita o`girilishi mumkin. Bilvosita o`girish avtomatik bo`ladi, buni kompilyatornig o`zi bajaradi. Bevosita o`girish qiymatni boshqa tipga o`girishda sodir bo`ladi. Bilvosita o`girish ma`lumotlar yo`qotilishini oldini oladi. Masalan short (2 bayt) tipidan int (4 bayt) tipiga qiymatni bilvosita o`girish mumkin. Bunda short tipidagi qiymat qanday bo`lishidan qat`iy nazar u int tipiga o`girilayotganda yo`qotilmaydi:

```
short  x = 1;
int  y = x; //bilvosita o`girish
```

Agar shu ishning teskarisi amalga oshirilayotgan bo`lsa, u kompilyator tomonidan bajarilmaydi. Int tipidan short tipiga bilvosita o`girilmaydi:

```
short x;
int y = 5;
x = y; //kompilyasiya qilinmaydi.
```

Bu amalni bevosita o`girish orqali bajarish mumkin:

```
short x;
int y = 5;
x = (short) y; //kompilyasiya qilinadi.
```

O`zgaruvchilar

Dastur ishlash davomida ma`lumotlar vaqtinchalik saqlashiga to`g`ri keladi. Bu o`zgaruvchilar va o`zgarmaslar orqali amalga oshiriladi. O`zgaruvchilar aniq tip obyektining xotirada joylashishini unga murojaatni ta`minlaydi. Yuqoridagi misolda x va y o`zgaruvchilar. O`zgaruvchilar e`lon qilish vaqtida qiymatga ega bo`lishi, yoki dastur davomida qiymat o`zlashtirilishi mumkin. O`zgaruvchilarni e`lon qilish:

```
o`zgaruvchi tipi o`zgaruvchi nomi; yoki
o`zgaruvchi tipi o`zgaruvchi1 nomi, o`zgaruvchi2 nomi, …, o`zgaruvchiN nomi;
o`zgaruvchi tipi o`zgaruvchi nomi=qiymati;
```

o`zgaruvchi tipi o`zgaruvchi1 nomi=qiymati, o`zgaruvchi2 nomi=qiymati, …, o`zgaruvchiN nomi=qiymati;

O`zgaruvchilarga qiymat o`zlashtirish

Yangi o`garuvchi hosil qilish uchun avval uning tipini aniqash kerak yoki avvaldan mavjud tiplardan foydalanish kerak. O`zgaruvchini ishlatishdan oldin unga nom berish va uning tipini ko`rsatish lozim. Uni e`lon qilish vaqtida unga qiymat o`zlashtirish, yoki dastur bajarilishi davomida qiymat o`zlashtirish mumkun. Qiymat o`zlashtirish uchun “=” operatoridan foydalaniladi:

```
class Variables
{
 static void Main()
 {
  int myInt = 10; // e`lon qilish vaqtida qiymat o`zlashtirish
  System.Console.WriteLine ( "e'lon qilish vaqtida qiymat o'zlashtirilgandan keyin myInt: { 0 } ",myInt);
  myInt = 5; //dastur ishlash davomida qimat o`lashtirish
  System.Console.WriteLine("myInt ga qiymat o'zlashtirilgandan keyin: {0}", myInt);
 }
}
```

Dastur ishining natijasi quyidagicha:

```
e'lon qilish vaqtida qiymat o'zlashtirilgandan keyin: 10
myInt ga qiymat o'zlashtirilgandan keyin: 5
```

Har bir o`garuvchining qiymatini ishlatishdan oldin unga qiymat o`zlashtirish kerak. M:

```
class Variables
static void Main()
{
int myInt; //faqat e`lon qilish
System.Console.WriteLine ( "e'lon qilingandan keyin myInt: { 0 } ",myInt);
myInt = 5; //dastur ishlash davomida qimat o`lashtirish
System.Console.WriteLine("myInt ga qiymat o'zlashtirilgandan keyin: {0}", myInt);
}
```

Bu vaqtda kompilyator xatolik haqida xabar beradi.

O`zgarmaslar

O`zgarmaslar qiymati dastur ishlash davomida o`zgarmaydigan o`zgaruvchilar. Dastur ishlash davomida o`garuvchining qiymatini o`zgartirmasdan saqlashga to`g`ri keladi.Masalan Pi sonini o`zgarmas sifatida saqlash maqsadga muvofiq. Bu uchun Pi o`zgarmas sifatida e`lon qilinishi kerak:

```
const double Pi= 3.14159265358979323846264338327950;
```

C# da 3 turdagi o`zgarmaslar mavjud: literalli, simvolli va to`plamli o`zgarmaslar. Quyidagi misolni ko`ramiz:

```
x = 100;
```

100 qiymati bu literalli o`zgarmas. 100 ning qiymati hamma vaqt 100. 100 ga yangi qiymat o`zlashtirish mumkin emas. 100 ni hech qachon 55 qiymatini ko`rsatadigan qilish mumkin emas. Simvolli o`zgarmaslar o`zgarmas qiymatga nom beradi. Simvolli o`zgarmaslarni const kalit so`zi orqali e`lon qilinadi va quyidagi sintaksis qo`llaniladi:

```
const identifikator tipi = qiymat;
```

Har bir o`zgarmas e`lon qilish vaqtida qiymat o`zlashtirilishi shart va uning qiymati dastur bajarilishi davomida o`zgartirilmasligi kerak:

```
const double Pi= 3.14159265358979323846264338327950;
```

Pi simvolli o`zgarmas.

```
class Constants
{
 static void Main()
 {
  const int pi = 3.14159265358979323846264338327950;
  const int g = 9.81; //gravitasiya doimiysi
  System.Console.WriteLine{"Pi soni: {0}" , pi);
  Syseem.Console.WriteLine("gravitasiya doimiysi: {0}", g);
 }
}
```

Dastur ishining natijasi quyidagicha:

```
Pi soni: 3.14159265358979323846264338327950
gravitasiya doimiysi: 9.81
```

Simvolli o`zgarmaslar literalli o`zgarmaslar kabi bir vazifani bajaradi faqat ulardan farqi u nomga ega bo`lib, nomidan qanday maqsadda ishlatilayotganini aniqlash mumkin.

To`plam o`zgarmaslar

To`plam o`zgarmaslar kuchli alternativ o`zgarmaslar hisoblanadi. U bir nechta nomlangan o`zgarmaslardan iborat. Faraz qilay bizda bir necha kishinig tug`ilgan yili haqida ma`lumot bor. Dasturda ularni o`zgarmas sifatida e`lon qilish uchun quyidagicha yozish kerak:

```
const int maryBirthday = 1955;
const int ivanBirthday = 1980;
const int pavelBirthday = 1976;
```

O`zaro bog`lanmagan 3 ta o`zgarmas hosil bo`ldi. Ular orasida mantiqiy bog`lanishni hosil qilish uchun to`plam o`zgarmaslardan foydalaniladi. U quyidagicha bo`ladi:

```
enum FriendsBirthday
{
maryBirthday = 1955;
ivanBirthday = 1980;
pavelBirthday = 1976;
}
```

Har bir to`plam o`zgarmasning o`z tipi bo`ladi. Bu tiplar butun sonli tiplar(int, short, long,…), char tipi bulardan mustasno. Bu ko`rsatilgan tip har bir o`zgarmas uchun tegishli bo`ladi. Agar tip ko`rsatilmagan bo`lsa, unda int deb qabul qilinadi. Yuqoridagi misolda shunday qilingan.

```
enum FriendsBirthday : long
{
maryBirthday = 1955;
ivanBirthday = 1980;
pavelBirthday = 1976;
}
```

Agar to`plam o`zgarmasning o`zgarmaslai qiymati ko`rsatilmagan bo`lsa, unda ular mos ravishda 0 dan boshlab qiymat qabul qiladi va qiymat bittadan oshib boradi:

enum Days {Sat, Sun, Mon, Tue, Wed, Thu, Fri};

Bunda Sat=0, Sun=1,…, Fri=6 qiymat qabul qiladi.

Agarda enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri}; bo`lsa, unda Sat=1, Sun=2,…, Fri=7 qiymat qabul qiladi.

Agarda enum Days {Sat=1, Sun, Mon=10, Tue, Wed, Thu, Fri}; bo`lsa, unda Sat=1, Sun=2, Mon=10,Tue=11,…, Fri=14 qiymat qabul qiladi.

To`plam o`zgarmasning umumiy ko`rinishi:

[*attribut*] [*modifikator*] enum *identifikator* [:tip] {o`zgarmaslar ro`yxati } [;]

[] qavs bo`lishi yoki bo`lmasligi ham mumkin degan ma`noni bildiradi.
To`plam o`zgarmasga murojaat quyidagicha amalga oshiriladi:
Days.Sat yani identifikator.o`zgarmas
To`g`ridan-to`g`ri o`zgarmasning qiymatini olish mumkin emas. Bu uchun avval enum tipini butun tipga o`girish kerak:

```
using System;
public class EnumTest
{
  enum Range :long {Max = 2147483648L, Min = 255L};
  public static void Main()
  {
    long x = (long) Range.Max;
    long y = (long) Range.Min;
    Console.WriteLine("Max = {0}", x);
    Console.WriteLine("Min = {0}", y);
  }
}
```

Natija:

```
Max = 2147483648
Min = 255
```

Modifikator va atributlar haqida keyin to`xtalamiz.

Satrli o`zgarmaslar

Satrli o`zgarmasni e`lon qilish uchun satrni ("My string") ikkitalik qo`htirnoqqa olish kerak. Buni o`zgaruvchilarga qiymat berishda, funksiyaga parametr sifatida berishda foydalanish mumkin:

```
System.Console.WriteLine("Pi soning qiymati: {0}", Pi);
string mystring="Hello";
```

Operatorlar

C# da keng operatorlar to`plami mavjud. Ular dasturchiga turli ifodalarni hosil qilish, ifodalarni hisoblash uchun kuchli boshqaruv imkoniyatini beradi. Operatorlar 4 ta katta guruhga bo`linadi: arifmetik, razryadli, mantiqiy va munosabat operatorlari. C# da maxsus holatlarni

boshqarish uchun ham operatorlar mavjud, ulani keyinchalik ko`rib chiqamiz.

Arifmetik operatorlar

C# da quyidagi arifmetik amallar aniqlangan:

Operator	***Vazifasi***
+	Qo`shish
-	Ayirish, - minus
*	Ko`paytirish
/	Bo`lish
%	Bo`linmaning qoldig`ini o`lish
--	Dekremint
++	Inkrement

+, -, *, / amallari C# da ham boshqa tillardagi kabi mos ravishda qo`shish, ayirish, ko`paytirish, bo`lish vazifasini bajaradi.Lekin / operatorida ozgina farq bor. Butun sonlarni bo`lishda qoldiq tashlab yuboriladi. 10/3 amalidan keyin natija 3 ga teng bo`ladi. Uning qoldig`i % operatori orqali olinadi. 10%3 = 1. C# da % operatorini butun tiplarga ham haqiqiy tiplarga qo`llash mumkin. 10.0%3.0 = 1 bo`ladi. M:

```
using System;
class ModDemo
{
  public static void Main()
  {
    int iresult, irem;
    double dresult, drem;
    iresult = 10 / 3;
    irem = 10 % 3;
    dresult = 10.0 / 3.0;
    drem = 10.0 % 3.0;
    Console.WriteLine( "10/3 ning natijasi va qoldig`i: " + iresult + " " + irem);
    Console.WriteLine( "10.0/3.0 ning natijasi va qoldig`i:" +dresult + " " + drem);
   }
}
```

Natija:

```
10/3 ning natijasi va qoldig`i: 3 1
10.0/3.0 ning natijasi va qoldig`i: 3.33333333333333 1
```

Inkrement va dekrement

Inkrement(++) va dekrement(--) operatorlari mos ravishda operand qiymatini bittaga oshiradi, kamaytiradi.

x=x+1; //x ning qiymatini 1 taga oshirish

x++; //x ning qiymatini inkrement operatori orqali bittaga oshirish

Bu ikki usul ham teng kuchli.

x=x-1; //x ning qiymatini 1 taga kamaytirish

x--; //x ning qiymatini dekrement operatori orqali bittaga kamaytirish

Inkrement va dekrement operatorlari operandning oldida ham bo`lishi mumkin.

++x; // inkrement operatorinig prefiksli ko`rinishi

--x; // dekrement operatorinig prefiksli ko`rinishi

x++; // inkrement operatorinig postfiksli ko`rinishi

x--; // dekrement operatorinig postfiksli ko`rinishi

Ularning bir-biridan farqini quyidagi misolda ko`rish mumkin:

int x=10;

int y=x++;

Bunda y ning qiymati 10 ga teng, x ning qiymati 11 ga teng bo`ladi. Yani bunda x ning qiymati avval y ga o`zlashtirilib, so`ng inkrement amali bajariladi.

int x=10;

int y=++x;

Bunda y ning qiymati 11 ga teng, x ning qiymati 11 ga teng bo`ladi. Yani bunda avval inkrement amali bajarilib, so`ng x ning qiymati y ga o`zlashtiriladi. Dekrement operatoriga ham shu qoida tegishli. Misol:

```
using System;
class PrePostDemo
{
  public static void Main()
  {
    int x, y;
    int i;
    x = 1;
    Console.WriteLine("y = x + x++; instruksiyasi orqali tuzilgan qator");
    for(i = 0; i < 10; i++)
    {
      y = x + x++; // ++ operatorining postfiksli ko`rinishi
      Console.WriteLine(y + " " ) ;
    }
```

```
    Console.WriteLine();
    x = 1;
    Console.WriteLine("y = x + ++x; instruksiyasi orqali tuzilgan qator");
    for ( i = 0; i < 10; i++)
    {
      y = x + ++x; // ++ operatorining prefiksli ko`rinishi
      Console.WriteLine(y + " " ) ;
     }
     Console.WriteLine();
  }
}
```

Natija:

```
y = x + x++; instruksiyasi orqali tuzilgan qator
2
4
6
8
10
12
14
16
18
20
y = x + ++x; instruksiyasi orqali tuzilgan qator
3
5
7
9
11
13
15
17
19
21
```

Munosabat va mantiqiy operatorlar

Munosabat operatorlari ikkilik sistema(chin/yolg`on) orqali baholanadi. Yani ulaning natijasi chin yoki yolg`on bo`ladi. Mantiqiy operatorlar ko`pincha munosabat operatorlari bilan birga qo`llaniladi.

Mantiqiy va munosabat operatorlari operandlarni taqqoslash, shartli operatordagi shartni hosil qilishda foydalaniladi. Mantiqiy va munosabat operatorlaridan tuzilgan ifoda natijasi, oprandlar qiymati hamma vaqt chin(true) yoki yolg`on(false) bo`ladi.
Munosabat operatorlari: tenglik (= =), teng emas (!=), kichik (<), kichik yoki teng (<=), katta (>) va katta yoki teng(>=). Tenglik (= =) va teng emas(!=) operatorlari har qanday tipga qo`llanilsa bo`ladi. Qolgan kichik (<), kichik yoki teng (<=), katta (>) va katta yoki teng(>=) operatorlari tartiblash xususiyatiga ega bo`lgan tiplarda qo`llaniladi. Munosabat operatorlar arifmetik operatorlarga nisbatan quyi prioritetga ega. Masalan 2 + 3 < 2 * 9 ifoda (2 + 3) < (2*9) ifodaga teng kuchli. Quyida mntiqiy operatolar yuqori prioritetdan quyi prioritetga qarab tartiblangan:
Not, And, Or, Xor, Eqv and Imp
Mantiqiy operatorlar munosabat operatorlariga nisbatan quyi prioritetga ega. 2 < 3 And 4 >= -1 ifoda (2 < 3) And (4 >= -1) ifodaga teng kuchli.

Razryadli operatorlar

Razryadli operatorlar: &, <<, >>, ^, |.
& operatorini ham unar ham binar operator sifatida qo`llaniladi.
& operand
operand & operand
Unar & operatori o`zining operandining adresini qaytaradi.
Binar & operatori butun tip va mantiqiy tiplar uchun qo`llaniladi. Matiqiy tipda qo`llanilganda mantiqiy konyunksiya amalini bajaradi. Butun tipga qo`llanilganda, ikki operandning ikkilik sanoq sistemasidagi ifodalarining mos razryadlari orasida mantiqiy konyunksiya amalini qo`llaydi. Masalan:
7 & 10
$7_{10}=111_2$
$10_{10}=1010_2$
0111_2 & 1010_2 = 0010_2 yani mos razryadlar orasida konyunksiya amalini qo`llaymiz.
$0010_2=2_{10}$
Bundan kelib chiqadiki, 7 & 10 = 2 bo`ladi.

<< operatori(Chapga surish operatori) o`zining birinchi operandining razryadlarini ikkinchi operandi miqdoricha chapga suradi. Masalan:
1 << 3
$1_{10}=1_2$
1 ni bir marta chapga surganda 10 bo`ladi.
1 ni ikki marta chapga surganda 100 bo`ladi.
1 ni uch marta chapga surganda 1000 bo`ladi.
$1000_2=8_{10}$
Demak 1 << 3 = 8.
Yana bir misol:
5 << 1
$5_{10}=101_2$
101 ni 1 marta chapga surganda 1010 bo`ladi.
$1010_2=10_{10}$
Demak 5 << 1 = 10.

>> operatori(o`ngga surish operatori) o`zining birinchi operandining qiymatini ikkinchi operandi miqdoricha o`ngga suradi. Masalan:
5 >> 1
$5_{10}=101_2$
101 ni 1 marta o`ngga surganda 10 bo`ladi.
$10_2=2_{10}$
Demak 5 >> 1 = 2.

^ operatori mantiqiy(bool) va butun tiplarda qo`llaniladi. Bu operator razryadlar orasida quyidagi amalni bajaradi. Natija 1 yoki true bo`lishi uchun razryadlardan faqat bittasi bir (butun tip uchun) yoki true (mantiqiy tip uchun) qiymatga (yani ular bir-biridan farq qilishi kerak) teng bo`lishi kerak, qolgan hollarda natija 0 yoki false bo`ladi:
5 ^ 7
$5_{10}=101_2$
$7_{10}=111_2$
$101_2 \text{ ^ } 111_2 = 010_2$
$010_2=2_{10}$
Demak 5 ^ 7 = 2.

Misol:

```
using System;
class Test
{
  public static void Main()
  {
    Console.WriteLine(true ^ false);  // logical exclusive-or
    Console.WriteLine(false ^ false); // logical exclusive-or
    Console.WriteLine("0x{0:x}", 0xf8 ^ 0x3f);   // bitwise exclusive-or
  }
}
```

Natija:

```
True
False
0xc7
```

| operatori mantiqiy(bool) va butun tiplarda qo`llaniladi. Bu operator razryadlar orasida mantiqiy dizyunksiya amalini bajaradi. Natija 0 yoki false bo`lishi uchun razryadlarning ikkalasi ham 0 (butun tip uchun) yoki false (mantiqiy tip uchun) qiymatga teng bo`lishi kerak, qolgan hollarda natija 1 yoki true bo`ladi:

5 | 4

$5_{10}=101_2$

$4_{10}=100_2$

$101_2 \mid 100_2 = 101_2$

$101_2=5_{10}$

Demak 5 | 4 = 5.

Quyidagi jadvalda razryadli operatorlar ustida amallar keltirilgan:

p	q	p & q	p | q	p ^ q	~p
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	1	0	0

Mantiqiy && operatori mantiqiy and operatoriga sinonim yani bir vazifani bajaradi. && operatori razryadli & operatorlari mantiqiy(bool) tip ustida bir xil amal bajaradi. Farqi & operatori birinchi operand false qiymatga ega

bo`lganda, ikkinchi operandni tekshirib o`tirmaydi. Chunki kamida bittasi false bo`lsa, konyunksiya false bo`ladi.
Mantiqiy || operatori mantiqiy and operatoriga sinonim yani bir vazifani bajaradi. || operatori razryadli | operatorlari mantiqiy(bool) tip ustida bir xil amal bajaradi. Farqi | operatori birinchi operand true qiymatga ega bo`lganda, ikkinchi operandni tekshirib o`tirmaydi. Chunki kamida bittasi true bo`lsa, dizyunksiya true bo`ladi.

Qiymat berish operatorlari

Qiymat berish operatori o`zgaruvchilarga qiymat berish uchun ishlatiladi. Qiymat berish operatorlari: =, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=.
= operatori o`ng tomondagi operand qiymatini chap tomondagi operandga o`zlashtiradi. Bunda ikki operand ham bir xil tipga tegishli bo`lishi kerak, aks holda o`ng tomondagi operand qiymatini chap tomondagi operand tipiga o`girib, keyin o`zlashtirish kerak. Misol:

```
using System;
class Test
{
  public static void Main()
  {
    double x;
    int i;
    i = 5; // int tipidagi o`zgaruvchiga int tipidagi qiymatni berish
    x = i; // double tipidagi o`zgaruvchiga int tipidagi o`zgaruvchi qiymatini berish
    i = (int)x; // double tipidagi o`zgaruvchi qiymatini int tipiga o`girib o`zlashtirish
Console.WriteLine("i is {0}, x is {1}", i, x);
    object obj = i;
    Console.WriteLine("boxed value = {0}, type is {1}", obj, obj.GetType());
    i = (int)obj;
    Console.WriteLine("unboxed: {0}", i);
  }
}
```

Natija:

```
i is 5, x is 5
boxed value = 5, type is System.Int32
unboxed: 5
```

+= operatori chap tomondagi operand qiymatini o`ng tomondagi operand qiymatiga qo`shib, so`ng yig`ndini chap tomon operandiga o`zlashtiradi.
Misol:

```
using System;
class Test
{
  public static void Main()
  {
    int a = 5;
    a += 6;
    Console.WriteLine(a);
    string s = "Hel";
    s += "lo";
    Console.WriteLine(s);
  }
}
```

Natija:
11
Hello

Qolgan *= /= %= &= |= ^= <<= >>= operatorlar yuqorida ko`rib o`tilganidek, chap va o`ng operandlar orasida mos ravishda *, /, %, &, |, ^, <<, >> amallarini bajarib, natijani chap operandga o`zlashtiradi.

```
using System;
class Test
{
  public static void Main()
  {
    int a = 5;
    a -= 6;
    Console.WriteLine(a);

    a = 5;
    a *= 6;
    Console.WriteLine(a);

    a = 6;
    a /= 5;
    Console.WriteLine(a);

    double b = 5;
```

```
        b /= 6;
        Console.WriteLine(b);

        a = 15;
        a %= 6;
        Console.WriteLine(a);

        a = 7;
        a &= 10;
        Console.WriteLine(a);

        a = 5;
        a |= 4;
        Console.WriteLine(a);

        a = 5;
        a &= 7;
        Console.WriteLine(a);

        a = 1;
        a <<= 3;
        Console.WriteLine(a);

        a = 5;
        a >>= 1;
        Console.WriteLine(a);


    }
}
```

Natija:

```
-1
30
1
0.833333333333333
3
2
5
2
8
2
```

Yana qiymat berishning quyidagi usuli ham mavjud:
a = b = c = d = 10;
Bunda bir ifodaning o`zida a, b, c, d o`zgaruvchilarning qiymati 10 ga tenglashtirilmoqda. Bunda a, b, c, d o`zgaruvchilar bir xil tipga ega bo`lishi kerak.

? operatori

? operatori C# da if-then-else konstruksiyasi o`rnida qo`llaniladi. ? operatori 3 ta ifoda bilan birga ishlatiladi. Ishlatilishi:
ifoda1 ? ifoda2: ifoda3;
ifoda1 bool tipiga tegishli bo`lishi kerak. ifoda2 va ifoda3 tiplari bir xil bo`lishi kerak. Agar ifoda1 ning qiymati true bo`lsa, ifoda2 bajarilib uning natijasi ? operatorining natijasi bo`ladi. Aks holda, ifoda3 bajarilib uning natijasi ? operatorining natijasi bo`ladi. Misol:

```
absval = val < 0 ? -val : val ; // absolyut qiymatni olish
```

Bu misolda agar val<0 bo`lsa, absval=-val bo`ladi, aks holda yani val>=0 bo`lsa, absval=val bo`ladi.

Probel va ( ) qavslarini ishlatish

C# da probellar dastur kodini o`qishni, tushunishni osonlashtiradi va chiroyli ko`rinishga keltiradi. Masalan:

```
x=10/y*(127/x);
x = 10 / y * (127 / x);
```

Bu ikki ifoda teng kuchli, ammo ikkinchisi tushunarliroq.
( ) qavslarni dastur kodida ishlatish operatorlarning prioritetini o`zgartiradi va tushunarli shaklga keltirishi mumkin. Yani qavs ichidagi amal birinchi bajariladi.Misol:

```
x = y/3-34*temp+127;
X = (y/3) - (34*temp) + 127;
```

Bu ikki ifoda teng kuchli, ammo ikkinchisida qavslar ishlatilgani uchun tushunarliroq.

Ifoda va instruksiya

Ifoda bu dastur kodining qatori hisoblanib, qiymatni aniqlaydi. Oddiy ifodaga misol:

```
myValue = 100;
```

Instruksiya bu dastur kodidagi tugallangan ifoda hisoblanadi. C# da tuzilgan dastur instruksiyalar ketma-ketligidan iborat. Har bir instruksiya ; (nuqta vergul) bilan tugatilishi kerak:

```
int x; // instruksiya
x = 10; // boshqa bitta instruksiya
int y = x; // yana bitta boshqa instruksiya
```

C# da yana instruksiyalar bloki mavjud. U bir qancha instruksiyalardan iborat bo`lib, {} (figurali qavslarga) olinadi.

```
{
 int x; // instruksiya
 x = 10; // boshqa bitta instruksiya
 int y = x; // yana bitta boshqa instruksiya
}
```

Misolda 3 ta instruksiya bitta instruksiyaning elementlari sifatida qaraladi.

Boshqaruv instruksiyalari

C# dastur kodlarining bajarish tartibini boshqarish uchun boshqaruv instruksiyalaridan foydalaniladi. 3 xil turdagi boshqaruv instruksiyalari mavjud: tanlash instruksiyalari(if, switch), iterasion instruksiyalar(for-, while-, do- va foreach sikllaridan iborat) va o`tish instruksiyalari(break, coninue, goto, return, throw).

if instruksiyasi

if instruksiyasi quyidagicha yoziladi:

```
if (shart) instruksiya;
else instruksiya;
```

Bu erda instruksiya sifatida C# tilining bitta instruksiyasi qaraladi. Else qismi shart emas. Instruksiya o`rnida instruksiyalar ketma-ketligi bo`lishi mumkin. U holda if instruksiyasi quyidagi ko`rinishga ega bo`ladi:

```
if (shart)
{
instruksiyalar ketma-ketligi
}
else
{
instruksiyalar ketma-ketligi
}
```

Bu erda shart shartli ifoda bo`lib, agar u bajarilsa, yani true qiymatga ega bo`lsa, if instruksiya bajariladi, aks holda else instruksiyasi bajariladi. if instruksiyasidagi shartli ifoda bool tipiga tegishli bo`lishi kerak. Misol:

```
using System;
class PosNeg
{
 public static void Main()
 {
  int i;
  for ( i = -5; i <= 5; i++)
  {
   Console.Write("Tekshirish " + i + ": ") ;
   if (i < 0) Console.WriteLine("son manfiy")
   else Console.WriteLine("son musbat");
  }
 }
}
```

Natija:

```
Tekshirish -5: Son manfiy
Tekshirish -4: Son manfiy
Tekshirish -3: Son manfiy
Tekshirish -2: Son manfiy
Tekshirish -1: Son manfiy
Tekshirish 0: Son musbat
Tekshirish 1: Son musbat
Tekshirish 2: Son musbat
Tekshirish 3: Son musbat
Tekshirish 4: Son musbat
```

Tekshirish 5: Son musbat

Bu misolda i o`zgaruvchining qiymati 0dan kichik bo`lganda if istruksiyasi bajarilib, natija sifatida sonning manfiyligi haqida ma`lumot berilgan, aks holda else instruksiyasi bajarilib, , natija sifatida sonning manfiyligi haqida ma`lumot chiqarilgan.

Ichma-ich joylashgan if instruksiyalari

Agar instruksiya sifatida yana boshqa if instruksiya qatnashgan bo`lsa, u holda bu ichma-ich joylashgan if instruksiyalari bo`ladi. Bu erda else instruksiyasi o`ziga yaqin oldingi hali biror else instruksiyasi bilan bog`lanmagan if instruksiyasiga tegishli bo`ladi. Misol:

```
if (i == 10)
{
 if (j < 20) a = b;
 if(k > 100) c = d;
 else a = c; // bu else-instruksiyasi if(k > 100) ga tegishli
}
else a = d; // bu else-instruksiyasi if(i == 10) ga tegishli
```

if – else – if konstruksiyasi

Bu konstruksiya dasturlashda ko`p qo`llaniladi. Uning yozilishi quyidagicha:

```
if (shart)
instruksiya;;
else if (shart)
instruksiya;
else if (shart)
instruksiya;
.
.
.
else
instruksiya;
```

Bu instruksiyaning ishlashi zinaga o`xshaydi. Yani shartlar ketma ket tekshirilaveradi, biror qatorda kelib, shart bajarilsa, unda if instruksiya bajariladi, zinaning qolgan qismi tashlab yuboriladi. Agar bironta ham shart bajarilmasa, unda oxirgi else instruksiyasi bajariladi. Agar bironta

shart ham bajarilmasa va oxirgi else instruksiyasi ko`rsatilmagan bo`lsa, unda hech qanday amal barilmaydi.

switch instruksiyasi

Switch tanlash instruksiyasi hisoblanadi. Bu instruksiya ko`p tarmoqli bo`lib, ma`lum bir to`plamdan elementni tanlash imkonini beradi. Bu amalni ichma-ich joytlashgan if instruksiyalari orqali ham bajarish mumkin. Lekin ko`pgina hollarda switch instruksiyasi effektli hisoblanadi. Bu instruksiyasi quyidagicha ishlaydi. Ifoda qiymati ro`yxatdagi o`zgarmaslar qiymati bilan navbatma-navbat taqqoslanadi. Agar qiymatlar teng bo`lsa, shunga mos instruksiya bajariladi. Switch instruksiyasining yozilishi quyidagicha:

```
switch (ifoda)
{
case o`zgarmas1:
instruksiyalar ketmaketligi
break;
case o`zgarmas2:
instruksiyalar ketmaketligi
break;
case o`zgarmas3:
instruksiyalar ketmaketligi
break;
.
.
.
default:
instruksiyalar ketmaketligi
break;
}
```

switch instruksiyasidagi ifoda butun tipga yoki string(satr) yoki char tipiga tegishli bo`lishi kerak. Bu erda ifoda sifatida odatda o`zgaruvchi ishlatiladi. case o`zgarmas sifatida literallar ishlatiladi. Literallarning tipi ifodaning tipi bilan bir xil bo`lishi kerak. Bitta switch instruksiyasida ikkita literalning qiymati bir xil bo`lmasligi kerak. Agar bironta ham case o`zgarmasning qiymati ifodaning qiymati bilan bir xil bo`lmasa, default instruksiyasidagi instruksiyalar bajariladi. switch instruksiyada default qismi shart emas. Misol:

```
using System;
class SwitchDemo
{
public static void Main()
{
int i;
for(i=0; I < 10; i++)
switch (i)
{
case 0:
Console.WriteLine("i nolga teng");
break;
case 1:
Console.WriteLine("i birga teng");
break;
case 2:
Console.WriteLine("i ikkiga teng");
break;
case 3:
Console.WriteLine("i uchga teng");
break;
case 4:
Console.WriteLine("i to\'rtga teng");
break;
default:
Console.WriteLine("i beshga teng yoki undan katta");
break;
```

Natija:

```
i nolga teng
i birga teng
i ikkiga teng.
i uchga teng.
i to'rtga teng.
i beshga teng yoki undan katta
i beshga teng yoki undan katta
i beshga teng yoki undan katta
i beshga teng yoki undan katta
i beshga teng yoki undan katta
```

Ichna-ich joylashgan switch instruksiyasi

C# da case instruksiyasi sifatida switch instruksiyasi qatnashgan bo`lsa, unda u ichma-ich joylashgan instruksiya deyiladi. Bunda tashqi case o`zgarmas va ichki case o`zgarmas bir xil qiymat qabul qilishi mumkin.
Misol:

```
switch (ch1)
{
case 'A':
Console.WriteLine("Bu A harfi tashqi switch instruksiyasi qismi");
switch (ch2)
{
case 'A':
Console.WriteLine("Bu A harfi ichki switch instruksiyasi qismi");
break;
case 'B': // ...
} // ichki switch instruksiyasi oxiri
break;
case 'B': // ...
```

for sikli

for sikli o`ziga tegishli instruksiyaning bir necha marta takroran bajarilishini ta`minlaydi. Bitta instruksiya uchun uning yozilishi quyidagicha:

```
for (e`lon qilish; shart; iterasiya) instruksiya;
```

Agar for sikli dastur bloki(bir nechta instruksiya ketma-ketligi)ni takroran bajarishi kerak bo`lsa, u holda for siklining ko`rinishi quyidagicha bo`ladi:

```
for (e`lon qilish; shart; iterasiya)
{
 instruksiyalar ketma-ketligi
}
```

E`lon qilish qismi qiymat o`zlashtirishdan iborat. Bu qismda siklni boshqaruvchi o`zgaruvhining boshlang`ich qiymati beriladi. Bu o`zgaruvchi hisoblagish vazifasini bajaradi, siklni boshqaradi. Shart qismi bool tipiga tegishli bo`lib, unda siklni boshqaruvchi o`zgaruvchi qiymati tekshiriladi. Shartning natijasi siklning yana bir bor bajarilishi yoki bajarilmasligini bildiradi. iterasiya ifoda bo`lib, u har bir sikldan so`ng siklni boshqaruvchi o`zgaruvchining qiymati qanday o`zgarishini

ko`rsatadi. for sikli toki shart true qiymatga ega ekan bajarilaveradi. Shart false qiymatga ega bo`lgandan keyin for siklidan keyingi instruksiya bajariladi. Misol:

```
using System;
class DecrFor
{
 public static void Main()
 {
  int x;
  for(x = 100; x > -100; x -= 5)
  Console.WriteLine(x);
 }
}
```

Shart hamma vaqt for sikli bajarilishidan oldin tekshiriladi. Yani agar shartni birinchi tekshirishdanoq false qiymat qabul qilsa, sikl umuman bajarilmaydi. Misol:

```
for(x = 10; x < 5; x++)
 Console.WriteLine(x); // bu instruksiya hech qachon bajarilmaydi.
```

for sikli odatda takrorlanishlar soni aniq bo`lgan holda ishlatiladi. Masalan quyidagi misolda 2 dan 20 gacha sonlar orasidan tub sonlarni ahiqlaydi. Agar son murakkab son bo`lsa, sonning eng katta bo`luvchisini chiqariladi:

```
using System;
class FindPrimes
{
 public static void Main()
{
 int num;
 int i;
 int factor;
 bool isprime;
 for (num = 2; num < 20; num++)
{
 isprime = true;
 factor = 0; // num  ning i ga qoldiqsiz bo`linishini aniqlaymiz.
 for(i=2; i <= num/2; i++) {
 if ((num % i) == 0)
{
 // agar num i ga qoldiqsiz bo`linsa, demak num murakkab son
 isprime = false;
 factor= i;
```

```
 if (isprime)
 Console.WriteLine(num + " - tub son");
 else
 Console.WriteLine(num + " ning eng katta bo\'luvchisi " + factor");
 Natija:
2 - tub son
3 - ub son
4 ning eng katta bo'luvchisi 2
5 - tub son
6 ning eng katta bo'luvchisi 3
7 - tub son
8 ning eng katta bo'luvchisi 4
9 ning eng katta bo'luvchisi 3
10 ning eng katta bo'luvchisi 5
11 - tub son
12 ning eng katta bo'luvchisi 6
13 - tub son
14 ning eng katta bo'luvchisi 7
15 ning eng katta bo'luvchisi 5
16 ning eng katta bo'luvchisi 8
17 - tub son
18 ning eng katta bo'luvchisi 9
19 - tub son
```

for siklining turli variantlari

for siklini bohqarish uchun bir nechta o`zgaruvchidan foydalanish mumkin. Bu holda e`lon qilish qismida har bir o`zgaruvchiga qiymat berish vergul bilan ajratiladi. Misol:

^отделяются запятыми. Вот пример:

```
// Использование запятых в цикле for.
using System;
class Comma
{
 public static void Main()
 {
 int i, j;
 for(i = 0, j = 10; i < j; i++, j--)
 Console.WriteLine("i va j: " + i + " " + j);
```

Natija:
i va j: 0 10
i va j: 1 9
i va j: 2 8
i va j: 3 7
i va j: 4 6
Bu dasturda vergul bilan e`lon qilish qismida i va j, iterasiya qismida i++ va j—iterasiyalari ajratilgan. Sikl boshida i va j  o`zgaruvchilarga boshlang`ich qiymat beriladi va har bir sikldan so`ng i ning qiymati bittaga oshiriladi, j ning qiymati 1 taga kamaytiriladi. E`lon qilish qismida va iterasiya qismida xohlagancha o`zgaruvchidan foydalanish mumkin.

Shartli ifoda
for siklidagi shartli ifoda bool tipidagi qiymat qaytaradigan har qanday ifoda bo`lishi mumkin. Quyidagi dasturda shartli ifoda sifatida done o`zgaruvchisi ishlatilmoqda:

```
using System;
class forDemo
{
 public static void Main() {
 int i, j;
 bool done = false;
 for(i=0, j=100; !done; i++, j --)
{
 if(i*i >= j) done = true ;
 Console.WriteLine("i, j : " + i + " " + j ) ;
}
}
}
```

Natija:
i, j: 0 100
i, j: 1 99
i, j: 2 98
i, j: 3 97
i, j: 4 96
i, j: 5 95

i, j: 6 94
i, j: 7 93
i, j: 8 92
i, j: 9 91
i, j: 10 90

Bu misolda done o`zgaruvchisi true qiymatga ega bo`lgandan keyin sikl ishini tugatadi. Agar i ning qiymati kvadrati j qiymatidan katta yoki teng bo`lsa, siklning ichida done o`zgaruvchiga true qiymat o`zlashtiriladi.

Siklni aniqlashda elementlarning tushirib qoldirilishi

Siklda har qanday element(e`lon qilish qismi, shart, iterasiya qismi) yoki birdaniga hammasi tushirib qoldirilishi mumkin. Misol:

```
using System;
class Empty
{
 public static void Main()
 {
  int i;
  for(i = 0; i < 10; )
  {
  Console.WriteLine ("№ " + i) ;
   i++; // siklni boshqaruvchi o`zgaruvchining qiymatini bittaga oshiramiz
  }
 }
}
```

Natija:

№ 0
№ 1
№ 2
№ 3
№ 4
№ 5
№ 6
№ 7
№ 8
№ 9

Quyidagi misolda for siklidan e`lon qilish qismi ham tushirib qoldirilgan:

```
using System;
class Empty2
{
```

```
 public static void Main()
 {
  int i ;
  i = 0; // e`lon qilish qismini tushirib qoldiramiz
  for(; i < 10; )
  {
   Console .WriteLine ("№ " + i) ;
   i++; // siklni boshqaruvchi o`zgaruvchining qiymatini bittaga oshiramiz
  }
 }
}
```

Natija:

№ 0
№ 1
№ 2
№ 3
№ 4
№ 5
№ 6
№ 7
№ 8
№ 9

Bu usuldan e`lon qilish qismidagi ifoda uzun va murakkab bo`lganda foydalanish qulay.

Cheksiz sikl

for siklidagi barcha elementlar tuchirib qoldirilganda cheksiz sikl hosil bo`ladi. Misol:

```
for (; ;)
{
// …
}
```

Bunday sikllar cheksiz davom etadi.

Tanasiz sikl

C# da for sikli tanasiz yani unga bog`liq instruksiyalarsiz ham bo`lishi mumkin. Ba`zan bunday sikllar juda qo`l keladi. Quyidagi misolda 1 dan 5 gacha sonlarning yig`indisini hisoblash uchun tanasiz sikldan foydalangan:

```
using System;
```

```
class Empty3
{
 public static void Main()
 {
  int i ;
  int sum = 0;
  // 1 dan 5 gacha sonlarni qo`shish
  for(i = 1; i <= 5; sum += i++) ;
  Console.WriteLine("Yigindi: " + sum);
 }
}
```

Natija:

Yigindi: 15

for siklidagi boshqaruvchi o`zgaruvchini e`lon qilish

Agar siklni boshqaruvchi o`zgaruvchi faqat shu sikl uchun kerak bo`lsa va sikl tanasidan tashqarida ishlatilmasa, u holda boshqaruvchi o`zgaruvchini quyidagicha e`lon qilinadi. Quyidagi misol 1 dan 5 gacha sonlarning ham yig`indisini ham faktorialni hisoblaydi:

```
using System;
class ForVar
{
 public static void Main()
 {
  int sum = 0;
  int fact = 1;
  // 1 dan 5 gacha sonlarning ham yig`indisini ham faktorialni hisoblaymiz
  for(int i = 1; i <= 5; i++)
  {
   sum += i; // i faqat sikl tanasida aniqlangan
   fact *= i;
  }
 // bu erda i o`zgaruvchi noma`lum
 Console.WriteLine("yigindi: " + sum);
 Console.WriteLine("faktorial: " + fact);
 }
}
```

Bu misolda i o`zgaruvchi sikl tanasida ko`rinish sohasida bo`lib, sikl tashqarisida ko`rinish sohasida bo`lmaydi yani sikl tugatilishi bilan avtomatik tarzda i o`zgaruvchi yo`qotiladi.

while sikli

while siklining yozilishi quyidagicha:

whille (shart) instruksiya;

Bunda instruksiya sifatida bitta instruksiya tushuniladi. Bir nechta instruksiya qatnashishi uchun while sikli quyidagicha yoziladi:

```
whille (shart)
{
instruksiyalar ketma-ketligi;
}
```

Shart bool tipiga tegishli ifoda bo`lib, siklni boshqaradi. Toki shart true qiymat qaytararkan sikl ishlaydi. Shartli ifoda false qiymat qaytarganda dastur bajarilishi sikldan keyingi instruksiyaga beriladi. Misol:

```
using System;
class WhileDemo
{
 public static void Main()
 {
  int num;
  int mag;
  num = 435679;
  mag = 0;
  Console.WriteLine("Son: " + num);
  while(num > 0)
  {
   mag++;
   num = num /10;
  }
  Console.WriteLine("raqamlar soni: " + mag);
 }
}
```

Natija:

```
Son: 435679
raqamlar soni: 6
```

while siklidagi shart for siklidagi shart kabi siklga kirishdan oldin tekshiriladi. Sikl tanasidagi instruksiyalarning bajarilishi shartli ifodaning

natijasiga bog`liq. Agar birinchi tekshirishdanoq shartli ifoda false qiymat qaytarsa, sikl tanasidagi instruksiyalar bir marta ham bajarilmaydi.

do-while sikli

do-while sikli for va while sikllaridan farqli ravishda shartni sikl boshida emas, balki sikl oxirida tekshiradi. Bundan kelib chiqadiki do-while sikli hech bo`lmaganda bir marta bajariladi. Uning yozilishi quyidagicha:

```
do
{
instruksiyalar ketma-ketligi
}
while (shart);
```

{} figurali qavslar do-while siklida shart bo`lmasada, lekin ularni ishlatish dastur kodini o`qishni osonlashtiradi va while siklidan farqlashga yordam beradi. do-while sikli toki shartli ifoda true qiymat qaytararkan bajarilaveradi. Quyidagi misol sonning raqamlarini teskari tartibda yozadi:

```
using System;
class DoWhileDemo
{
 public static void Main()
 {
  int num;
  int nextdigit;
  num = 198;
  Console.WriteLine("Son: " + num);
  Console.Write("Soning raqamlari teskari tartibda: " ) ;
  do
  {
   nextdigit = num % 10;
   Console.Write(nextdigit);
   num = num / 10;
  } while(num > 0) ;
  Console.WriteLine() ;
 }
}
```

Natija:

```
Son: 198
Soning raqamlari teskari tartibda: 891
```

Har bir iterasiyada sonning oxirgi raqami sonni 10 ga bo`lib qoldig`ini olish orqali aniqlanadi. Olingan qoldiq ekranga chop etiladi. Nisbat natijasi esa num o`zgaruvchida saqlanadi. Bu jarayon son yani num ning qiymati 0 ga teng bo`lguncha davom ettiriladi.

Sikldan chiqish uchun break instruksiyasidan foydalanish

break instruksiyasi orqali sikldan tezda chiqishni tashkil qilish mumkin. break instruksiyasi ishlatilganda sikldagi shart va sikl tanasidagi qolgan instruksiyalar bajarilishi to`xtatilib, siklning ishi tugatiladi va boshqaruv sikldan keyingi instruksiyaga beriladi. Misol:

```
using System;
class BreakDemo
{
 public static void Main()
 {
  // break instruksiyasidan foydalanish
  for(int i=-10; i <= 10; i++)
  {
   if ( i > 0) break; // i > 0 bo`lganda sikldan chiqish
   Console.Write(i + " " );
  }
  Console.WriteLine("Tamom!");
 }
}
```

Natija:

```
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 Tamom!
```

while, do-while instruksiyalarida ham break instruksiyasi sikldan chiqish uchun foydalaniladi va sikl tanasida yoziladi. break instruksiyasi faqat o`zi yozilgan siklga ta`sir etadi yani faqat shu sikl ishini to`xtadi. Siklda bir necha break instruksiyasi qo`llanilishi mumkin.

continue instruksiyasidan foydalanish

Sikldan tezda chiqish bilan birga siklning joriy iterasiyasini tugatib keyingi iterasiyasiga o`tish uchun continue instruksiyasi qo`llaniladi. continue

instruksiyasidan keyingi instruksiyalar bajarilmasdan keyingi iterasiyaga o`tiladi. Misol:

```
using System;
class ContDemo
{
 public static void Main()
 {
  // Выводим четные числа между 0 и 100.
  for(int i = 0; i <= 100; i++)
  {
   if ((i%2) != 0) continue; // keyingi iterasiyaga o`tish
   Console.WriteLine(i);
  }
 }
}
```

Bu misolda faqat juft sonlar chop etiladi. Toq sonni aniqlaganda oldinroq keyingi iterasiyaga o`tiladi. while va do-while instruksiyasida continue instruksiyasi boshqaruvni shartli ifodaga beriladi, for siklida esa continue instruksiyasidan keyin avval iterasiya ifodasi bajariladi undan keyin shartli ifoda bajariladi.

goto instruksiyasi

goto instruksiyasi C# da shartsiz o`tish uchun ishlatiladi. goto instruksiyasi bajarilgandan keyin boshqaruv nishon bilan ko`rsatilgan instruksiyaga beriladi. Bu instruksiya dasturchilar tomonidan ko`p ishlatilmaydi. U dasturning o`qilishini qiyinlashtiradi. goto instruksiyasini ishlatmasdan ham dasturni yozish mumkin. goto instruksiyasini qo`llash uchun nishon e`lon qilish talab qiladi. Nishon C# da identifikator bo`lib, undan keyin ikki nuqta qo`yiladi. goto instruksiyasi yordamida quyidagi siklni hosil qilish mumkin:

```
x = 1;
loop1:
x++;
if(x < 100) goto loop1;
```

goto instruksiyasini yana switch instruksiyasidagi case va default tarmoqlariga o`tish uchun foydalaniladi. Bunda goto instruksiyasi switch instruksiyasi tanasida bo`lishi kerak. Yani tashqaridan switch

instruksiyasidagi case va default tarmoqlariga murojaat qilish mumkin emas. Misol:

```
using System;
class SwitchGoto
{
 public static void Main()
 {
  for(int i=l; i < 5; i++)
  {
   switch(i)
   {
    case 1:
     Console.WriteLine("case 1 tarmog'i");
     goto case 3;
    case 2:
     Console.WriteLine("case 2 tarmog'i");
     goto case 1;
    case 3:
     Console.WriteLine("case 3 tarmog'i");
     goto default;
    default:
     Console.WriteLine("default tarmog'i");
     break;
   }
   Console.WriteLine();
  }
 // goto case 1; bu xato ya`ni tashqaridan switch instruksiyasidagi case va
//default tarmoqlarga murojaat qilish mumkin emas.
 }
}
```

Natija:

```
case 1 tarmog'i
case 3 tarmog'i
default tarmog'i
case 2 tarmog'i
case 1 tarmog'i
case 3 tarmog'i
default tarmog'i
case 3 tarmog'i
default tarmog'i
default tarmog'i
```

Ba`zan ichma-ich joylashgan instruksiyalardan chiqish uchun goto instruksiyasini qo`llash maqsadga muvofiq. Masalan:

```
using System;
class Use_goto
{
 public static void Main()
 {
  int i=0, j=0, k=0;
  for(i=0; i < 10; i++)
  {
   for(j=0; j < 10; j++ )
   {
    for(k=0; k < 10; k++)
    {
     Consple.WriteLine("i, j, k: " + i + " " +  j + " " + k) ;
     if(k == 3) goto stop;
    }
   }
  }
  stop:
  Console.WriteLine("Stop! i, j, k: " + i +" " + j + " " + k);
 }
}
```

Natija:

```
i, j, k: 0 0 0
i, j, k: 0 0 1
i, j, k: 0 0 2
i, j, k: 0 0 3
Stop! i, j, k: 0 0 3
```

Bu dasturda goto instruksiyani qo`llamasdan xuddi shunday natijani olish uchun 3 marta if va break instruksiyasini qo`llash kerak.

Sinflar