C# kalbos žinynas

1. KALBOS ABĖCĖLĖ

C# kalbos abėcėlę sudaro:

1. 26 lotynų abėcėlės didžiosios ir mažosios raidės

   A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z,

   a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z.

2. 10 arabiškų skaitmenų:

   0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

3. Specialūs simboliai:

   " , { } | [ ] ( ) + – * / \ ; : ' ? < > _ ! & # . ~ = % ^ $ @

Programos komentaruose ir simbolių eilutėse galima rašyti ne tik C# kalbos simbolius, bet ir visus kompiuterio klaviatūroje esančius simbolius.

2. VARDAI IR BAZINIAI ŽODŽIAI

Programoje gali būti daug vienodo tipo duomenų. Kad galėtume duomenis atskirti ir nurodyti, su kuriais iš jų reikia atlikti nurodytus veiksmus, duomenys yra įvardijami.

Vardas – tai nedaloma tam tikrų kalbos abėcėlės simbolių seka, naudojama konstantoms, tipams, kintamiesiems ir kitoms programavimo kalbos konstrukcijoms įvardinti.

Vardai sudaromi iš lotyniškų raidžių, pabraukimo ženklo _ ir arabiškų skaitmenų taip:

1. Pirmasis vardo simbolis turi būti raidė arba pabraukimo ženklas.
2. Tolimesni simboliai yra raidės, skaitmenys, pabraukimo ženklas.
3. Vartotojo sudarytas vardas negali sutapti su baziniu žodžiu.
4. Didžiosios ir mažosios raidės traktuojamos skirtingai.

Baziniai žodžiai vartojami operatoriams, duomenų tipams ir kitoms programavimo kalbos konstrukcijoms žymėti. Jie turi iš anksto apibrėžtą prasmę ir kitiems tikslams nenaudojami. Baziniai žodžiai dažniausiai yra anglų kalbos žodžiai ar jų sutrumpinimai. Jie rašomi mažosiomis raidėmis. Kad baziniai žodžiai būtų lengviau pastebimi, kompiuterio ekrane ir knygose jie spausdinami paryškintu arba kitos spalvos šriftu.

3. PROGRAMOS KOMENTARAI

Komentarai programose C# kalba gali būti užrašomi dvejopai:

1. Pradedami simbolių pora `/*` ir užbaigiami simbolių pora `*/`. Tarp nurodytų simbolių porų galima rašyti ne tik C# kalbos, bet ir kitus simbolius. Taip užrašomi kelių eilučių arba vienos eilutės komentarai.
2. Pradedami dviem įstrižaisiais brūkšneliais `//`, po jų rašomas tekstas iki eilutės pabaigos. Taip užrašomi tik vienos eilutės komentarai.

4. BAZINIAI DUOMENŲ TIPAI

C# kalboje duomenys skirstomi į reikšmės (*angl*. value) ir nuorodos (*angl*. reference) tipus.

Duomenys gali būti sveikieji ir realieji skaičiai, simboliai, tekstai ir pan. Kiekvienas duomenų tipas turi vardą.

Baziniais duomenų tipais laikomi paprastieji duomenys, kurių reikšmės yra atskiros, pavienės. Šie tipai turi bazinius vardus ir jų reikšmės yra nedalomos. Dažniausiai naudojami baziniai duomenų tipai:

```
int       // sveikųjų skaičių tipas
double    // realiųjų skaičių tipas
char      // simbolinis tipas
bool      // loginis tipas
```

Duomenų tipui atmintis neišskiriama, o išskiriama tik to tipo kintamajam ar konstantai. Sveikaskaitiniams tipams priklauso loginis, simbolinis ir sveikasis tipai. Simbolinio tipo `char` kintamieji įgyja kompiuterio simbolius atitinkančias reikšmes. `char` tipo reikšmėmis gali būti:

- lotyniškos raidės (`'a' – 'z', 'A' – 'Z'`),
- skaitmenys (`'0' – '9'`),
- kiti simboliai (`'?',';'` ir pan.),
- valdymo simboliai (`'\n', '\t', '\a'` ir pan.).

C# kalba turi loginį duomenų tipą `bool`, jo konstantos žymimos žodžiais `true` ir `false`.

5. KINTAMASIS, KINTAMOJO REIKŠMĖ

Kintamieji skirti duomenims atmintyje laikyti. Jų vardai sudaromi iš raidžių, pabraukimo simbolio ir skaitmenų, tačiau pirmas ženklas turi būti raidė arba pabraukimo simbolis. Kiekvienam kintamajam reikia nurodyti, kokio tipo duomenis jis turi atmintyje laikyti. Kintamojo aprašymo struktūra:

```
tipas kintamojo_vardas;
tipas kintamųjų_sąrašas;
```

čia `tipas` – tai kintamajame saugomos reikšmės tipas. Kintamojo tipą nustato programuotojas. Kintamųjų sąraše vienodo tipo kintamuosius galima grupuoti, tada kintamųjų vardai atskiriami kableliu. C# kalboje kintamuosius galima aprašyti bet kurioje programos vietoje.

Po kintamojo aprašo C# kalbos kompiliatorius kintamajam išskiria atminties dalį, tačiau nesirūpina, kas joje saugoma. Tuo turi pasirūpinti programuotojas. Kintamųjų aprašo sakinyje kintamiesiems galima suteikti pradines reikšmes taip: tipas kintamojo_vardas = pradinė_reikšmė;

Pradinių reikšmių suteikimo kintamiesiems pavyzdžiai:

```
double pi = 3.1415;
char a = 'A';
int k = 5/3;
bool status = false;
```

Jeigu yra keletas to paties tipo kintamųjų, tuomet galima juos išvardyti viename sakinyje, atskiriant vardus kableliu.

Kompiliatorius kiekvienam kintamajam kompiuterio atmintyje skiria tiek vietos, kiek reikia nurodyto tipo duomenims laikyti. Kintamieji reikšmes (duomenis) gauna duomenų įvedimo (skaitymo) sakiniais arba priskyrimo sakiniais.

6. OPERATORIAI

Operatorius – tai veiksmas, suformuojantis reikšmę, vadinamą rezultatu. Pradiniai operatorių duomenys vadinami **operandais**. Operandais gali būti konstantos, kintamieji, kreipiniai į funkcijas, reiškiniai.

Tame pačiame reiškinyje gali būti daug įvairių operatorių, todėl reikia žinoti kokia tvarka jas atlikti. Operatorių **prioritetas** – tai operatorių atlikimo eilės numeris. Reiškinyje veiksmų atlikimo tvarka valdoma prioritetais, asociatyvumo savybėmis ir lenktiniais skliaustais.

Aritmetiniai operatoriai atliekami su sveikosiomis ir realiosiomis reikšmėmis, išskyrus dalybos liekanos operatorių `%`, kuris atliekamas tik su sveikosiomis reikšmėmis. Aritmetinio operatoriaus rezultatas bus realusis, jei nors vienas operandas bus realusis. Vienodo prioriteto aritmetiniai operatoriai atliekami iš kairės į dešinę. Pavyzdžiui, įvykdžius programos fragmentą

```
double y;
int x = 2;
y = 2./3 + x * x -(2 * x - 5);
```

kintamasis y įgys reikšmę `5.66667`.

Palyginimo operatorių `==` ir `!=` operandais turi būti diskretūs dydžiai, bet neturėtų būti realieji skaičiai, nes jie yra apytiksliai skaičiai.

Palyginimo operatoriai `==` (lygu) ir `!=` (nelygu) turi žemesnį prioritetą negu `< <= > >=` palyginimo operatoriai. Pavyzdžiui, įvykdžius programos fragmentą:

```
bool log;
x = 2;
log = x > 2 && x != 0;
```

kintamasis `log` įgys reikšmę `false`.

Priskyrimo operatorius yra =. Šis operatorius nurodo, kad reikšmė, esanti dešinėje ženklo pusėje yra priskiriama kintamajam, kurio vardas rašomas kairėje ženklo pusėje. Priskyrimo sakinio sintaksė:

```
kintamasis operatorius = reiškinys
```

Priskyrimo operatoriaus taikymo pavyzdžiai:

```
i = i*(i1 + i2);
x = x + y;
```

Operatoriai skliausteliai. Lenkti skliausteliai `()` naudojami operatorių, parašytų tarp lenktų skliaustelių, prioritetui padidinti, funkcijoms aprašyti bei iškviesti.

Kartais vieną duomenų tipą reikia pakeisti kitu (konvertuoti) išsaugant beveik tą pačią reikšmę. Tai atliekama **tipo keitimo operatoriumi**. Šį veiksmą galima atlikti keliais operatoriais, kurių sintaksės yra:

```
(tipas) operandas
tipas(operandas)
```

Operandu gali būti kintamasis, konstanta, reiškinys. Tipų keitimus reikia taikyti atidžiai, nes gali pasikeisti pradinės skaitinės reikšmės.

7. STANDARTINĖS MATEMATINĖS FUNKCIJOS

C# kalbos bibliotekose yra daug standartinių matematinių funkcijų. 1 lentelėje pateikti dažniausiai naudojamų funkcijų vardai ir trumpa jų paskirtis (plačiau apie funkcijų parametrų ir grąžinamų reikšmių tipus žiūrėkite programavimo terpės žinyne).

1 lentelė. Pagrindinės standartinės matematinės funkcijos

Funkcija	Paskirtis
`Abs(x)`	x absoliutus dydis
`Cos(x)`	kosinusas
`Exp(x)`	eksponentė
`Pow(x,y)`	x kelia y laipsniu
`Sin(x)`	sinusas
`Sqrt(x)`	kvadratinė šaknis
`Tan(x)`	tangentas

Standartinių matematinių funkcijų reikšmėms skaičiuoti naudojamos funkcijos, į kurias kreipinys užrašomas taip: `Math.FunkcijosVardas(argumentas/ai);`

Pavyzdžiui: apskaičiuoti, kiek bus 2 pakelti 8 (2^8)?

```
double r = Math.Pow(2, 8);
Console.WriteLine(„{0}“, r);
```

8. REIŠKINIAI IR JŲ REIKŠMIŲ SKAIČIAVIMAS

C# kalbos programose duomenų pertvarkymo veiksmai užrašomi reiškiniais. Matematikoje ir programavime reiškinio sąvoka suprantama vienodai.

Reiškinys yra kalbos konstrukcija naujoms reikšmėms formuoti. **Reiškiniu** vadinama operatorių, operandų ir lenktų skliaustų seka, kurioje atlikus nurodytus veiksmus gaunamas apibrėžto tipo rezultatas.

Paprasčiausiu reiškiniu yra konstanta, kintamasis, kreipinys į funkciją. Paprastieji reiškiniai turi operatoriaus ženklus ir operandus. Reiškinių pavyzdžiai:

```
12
x
a + b
12.3 + x
3 == 5
(2>3) || false
(3 < 4) && (-3 > 1)
```

Reiškinio rezultato tipas priklauso nuo jį sudarančių operandų ir operatorių tipų.

Reiškiniai gali būti: aritmetiniai (veiksmai aprašomi aritmetiniais operatoriais), loginiai (veiksmai aprašomi loginiais operatoriais), palyginimo (veiksmai aprašomi palyginimo operatoriais) ir pan. Dažnai reiškiniuose naudojome kelių tipų operatorius.

Paprasčiausi loginiai reiškiniai gaunami palyginimo operatoriumi sujungus du aritmetinius reiškinius. Sudėtingesni loginiai reiškiniai gaunami jungiant loginius reiškinius loginiais operatoriais:

- `&&` (IR),
- `||` (ARBA),
- `!` (NE).

Loginiai reiškiniai gali įgyti tik dvi reikšmes: `true` ir `false`.

Rašant programas C# kalba būtina žinoti sudėtingesnių reiškinių reikšmių skaičiavimo pagrindines taisykles, priešingu atveju galima gauti neteisingus skaičiavimo rezultatus. Veiksmų atlikimo tvarka reiškiniuose yra įprasta: pirmiausiai veiksmai atliekami skliaustuose, po to – pagal veiksmų prioritetus. Prioritetais apibrėžtą veiksmų atlikimo eilę galima pakeisti lenktais skliaustais.

Išvardinsime kelis reiškinių reikšmių skaičiavimo ypatumus:

1. Operatorių operandai turi atitikti apibrėžtus duomenų tipus.
2. Jei reiškinyje yra kelių tipų duomenys, tai kompiliatorius žemesnio tipo reikšmes transformuoja į aukštesnį tipą. Aritmetinio reiškinio atsakymas bus realusis, jei nors vienas operandas bus realusis.
3. Jeigu dalybos operatoriaus abu operandai yra sveikieji skaičiai, tai rezultatas yra sveikasis skaičius, kitais atvejais – realusis skaičius.
4. Aritmetiniai operatoriai turi aukštesnį prioritetą negu palyginimo operatoriai.
5. Jei reiškinyje yra keli vienodo prioriteto operatoriai ir nėra skliaustelių, tai aritmetiniai, palyginimo, loginiai, postūmio, bitų laukams atliekami iš kairės į dešinę, kitos (priskyrimo, sąlygos ir kt.) – iš dešinės į kairę.
6. Praktikoje dažnai sumaišomas priskyrimo operatoriaus ženklas `=` su operatoriaus lygu ženklu `==`.
7. Realieji skaičiai verčiami į sveikuosius skaičius atmetant jų trupmeninę dalį.

9. PRISKYRIMO SAKINYS

Priskyrimo sakinys – tai pats paprasčiausias iš vykdomų sakinių, kurio sintaksė yra tokia:

```
x = reiškinys;
```

čia ženklas `=` žymi priskyrimo operatorių. Įvykdžius sakinį, kairėje pusėje esančiam kintamajam `x` priskiriama dešinėje pusėje esančio reiškinio reikšmė. Reiškinys gali būti sudarytas ir iš vienos konstantos, ir vieno kintamojo. Kintamuoju `x` gali būti paprastas ar indeksuotas kintamasis (masyvo elementas). Reikia prisiminti, kai tam pačiam kintamajam priskiriama nauja reikšmė, ankstesnė jo reikšmė yra ištrinama. C# kalboje galima rašyti iš eilės kelis priskyrimus, bet būtina, kad kiekvieno priskyrimo sakinio kairėje pusėje (iki kito priskyrimo sakinio arba skliaustų) būtų tik kintamasis, o ne reiškinys. Viename reiškinyje kelių surašytų priskyrimo sakinių vykdymas pradedamas iš dešinės.

Kelių priskyrimo sakinių pavyzdys: `c = x = d = 4.0 + 2.4;`

čia pirmiausiai priskiriama `d = 6.4`, po to `x = 6.4;` toliau `c = 6.4`.

10. SĄLYGOS SAKINYS

Jei programoje nėra specialių nurodymų, tai jos sakiniai vykdomi nuosekliai ta tvarka, kuria jie surašyti. Tai natūrali programos vykdymo tvarka, kuri gali būti pakeičiama tik specialiomis valdymo struktūromis. Šiam tikslui vartojamos dvi pagrindinės valdymo struktūros: sąlygos ir išrinkimo sakiniai.

Sąlygos sakiniuose nurodomos sąlygos, nuo kurių reikšmių priklauso tolesnis vykdomų veiksmų parinkimas.

C# kalboje **sąlygos sakinys** yra dviejų alternatyvų konstrukcija, kuri išrenka vieną sakinį iš dviejų sakinių ir jį įvykdo. Šio sakinio sintaksė yra tokia:

```
if (sąlyga)
   sakinys1;
else
   sakinys2;
```

C# kalboje sąlyga būtinai rašoma skliausteliuose, ja gali būti aritmetinis, loginis, palyginimo reiškinys. Žodelis `else` naudojamas tik sakinio `if` viduje. Sąlygos sakinio veikimas: jeigu sąlyga yra teisinga, tai vykdomas `sakinys1`, jeigu ne – sakinys, einantis po žodelio `else`, t. y. `sakinys2`.

Pagal C# kalbos taisykles po sąlygos ir žodžio `else` galima rašyti tik vieną sakinį, todėl jeigu reikia vykdyti daugiau negu vieną sakinį, tuomet juos jungiame į vieną sudėtinį sakinį, panaudoję figūrinius skliaustelius `{ }`:

```
if (sąlyga)
{
   sakiniai;
}
else
{
   sakiniai;
}
```

Sakinys `if` gali turėti ir vieną šaką, kuri vykdoma, kai sąlyga yra teisinga, t. y. žodelio `else` gali nebūti. Šiuo atveju turime vieno varianto sakinį: `if (sąlyga) sakinys;`

Sakinyje `if` gali būti atliekami veiksmai, kai sąlygos reikšmė yra `false`, tada pirmojo sakinio vietoje rašomas tuščiasis sakinys arba tuščias sakinių blokas:

```
if (sąlyga)
   {}
else
    sakinys2;
```

arba

```
if (sąlyga)
   ;
else
    sakinys2;
```

C# kalbos sąlygos sakinio ypatumas yra tas, kad prieš žodelį `else` rašomas kabliataškis.

Paprasčiausios sąlygos aprašomos palyginimo reiškiniais, kurių sintaksė tokia:

```
R1 PalyginimoOperacija R2
```

Čia `R1` ir `R2` – aritmetiniai reiškiniai, o `PalyginimoOperacija` gali būti žymima tokiais simboliais:

==	lygu	<	mažiau	<=	mažiau arba lygu
!=	nelygu	>	daugiau	>=	daugiau arba lygu

Jei reiškinių reikšmės tenkina palyginimo sąlygas, tai palyginimo operacijos rezultatui suteikiama loginė reikšmė `true`, o jei ne – `false`.

Pirmas pavyzdys:

```
if (a > 0)
    Console.WriteLine(„Teigiamas“);
else
    Console.WriteLine(„Neigiamas arba nulis“);
```

Antras pavyzdys:

```
if (a > 0)
    Console.WriteLine(„Teigiamas“);
else if (a == 0)
         Console.WriteLine(„Nulis“);
     else
         Console.WriteLine(„Neigiamas“);
```

Sudėtingos sąlygos aprašomos loginiais reiškiniais, t. y. tokiais, kuriuose naudojamos loginės operacijos `&&` (IR), `||` (ARBA), `!` (NE). Jomis palyginimo reiškiniai jungiami į sudėtingesnius.

Loginių operacijų teisingumo lentelės:

Kintamųjų reikšmės		Operacijų su loginėmis reikšmėmis rezultatai		
a	b	a && b	a || b	! a
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

11. CIKLO SAKINYS `WHILE`

Ciklo sakinys `while` aprašo sąlyginį nežinomo kartojimų skaičiaus ciklą, kurio sintaksė yra tokia:

```
while (cikloVykdymoSąlyga)KartojamasSakinys;
```

`ciklo_Vykdymo_Sąlyga` – tai loginis reiškinys, kurio kintamiesiems prieš ciklą `while` turi būti suteiktos pradinės reikšmės. Jos turi būti keičiamos kartojamo sakinio viduje taip, kad vykdymo sąlyga po tam tikro kartojimų skaičiaus būtų netenkinama. Neįvykdžius šios sąlygos, gaunamas begalinis ciklas. `KartojamasSakinys` – tai C# kalbos bet koks sakinys (gali būti ir ciklo sakinys).

Ciklo veikimas: pirmiausiai tikrinama vykdymo sąlyga. Jeigu ji neteisinga, tai po sąlygos esantis sakinys nevykdomas nė karto, ciklas nutraukiamas. Jeigu sąlyga yra teisinga, tai vykdomas sakinys esantis po sąlygos, keičiama ciklo kintamojo reikšmė ir vėl grįžtama prie sąlygos tikrinimo. Šie veiksmai vykdomi tol, kol vykdymo sąlyga yra tenkinama. Vykdymo sąlyga gali būti aritmetinis reiškinys, bet tada ciklas vykdomas tol, kol reiškinio reikšmė nelygi nuliui.

Ciklo `while` antraštė valdo tik vieno sakinio kartojimą. Jeigu reikia kartoti kelis sakinius, tai jie jungiami į sudėtinį sakinį figūriniais skliausteliais `{}`:

```
while (ciklo_Vykdymo_Sąlyga)
{
    kartojami_Sakiniai;
}
```

Pavyzdys:

```
// Sveikojo skaičiaus skaitmenų išskyrimas:
int x;
 Console.WriteLine(„Įveskite sveikąjį skaičių:“);
 x = int.Parse(Console.ReadLine());
 Console.WriteLine(„Skaičiaus {0} skaitmenys: „, x);
 while (x > 0)
 {
     int d = x % 10;
     x = x / 10;
     Console.WriteLine(„{0}“, d);
 }
```

Įvedę skaičių x = 654321 ir įvykdę ciklą ekrane matysime skaičiaus skaitmenis:

```
1
2
3
4
5
6
```

12. CIKLO SAKINYS FOR

Dažnai turime atlikti veiksmus, iš anksto žinodami, jų kartojimo skaičių, pavyzdžiui dirbant su masyvais. Šiai situacijai aprašyti geriausiai tinka ciklas `for`, kuris įgalina organizuoti žinomo kartojimų skaičiaus ciklą. Ciklo `for` sintaksė yra tokia:

```
for (i1; i2; i3) kartojamas_Sakinys;
```

Šiame sakinyje:

- reiškiniai `i1` ir `i3` gali būti sudaryti iš kelių sakinių;
- reiškinys `i1` apibrėžia veiksmus, kurie turi būti įvykdomi iki ciklo pradžios, vykdomas vieną kartą. Paprastai jis naudojamas ciklo kintamojo (kintamasis, kuris vartojamas ciklo pabaigai apibrėžti) pradinei reikšmei nustatyti;
- vykdymo sąlyga `i2` dažniausiai yra loginis ar aritmetinis reiškinys, kuris apibrėžia ciklo pabaigą. Jeigu ciklo pradžioje sąlyga `i2` yra netenkinama, tai ciklas nevykdomas nė karto.
- reiškinys `i3` nusako ciklo kintamojo kitimą ir vykdomas kiekvieną kartą baigus vykdyti kartojamą sakinį. Įvykdžius `i3`, tikrinama sąlyga `i2`, ir viskas kartojama `...;`
- pradinė ir galutinė ciklo kintamojo reikšmės gali būti ne tik konstantos, bet ir kintamasis ar reiškinys;
- galima praleisti vieną ar visus ciklo antraštės parametrus, bet kabliataškiai yra būtini;
- po ciklo antraštės kabliataškis nededamas.

Cikle kartojamas sakinys gali būti ir sudėtinis, kuris užrašomas figūriniuose skliausteliuose `{}`:

```
for (i1; i2; i3) {kartojamiSakiniai;}
```

Ciklo `for` veikimas: skaičiuojams `i1` reiškinys, po to tikrinama, ar tenkinama sąlyga `i2`, jei netenkinama – vykdomas kartojamas sakinys, po to skaičiuojamas reiškinys `i3` ir vėl tikrinama sąlyga `i2`, ir t. t. tol, kol sąlyga `i2` tenkinama.

Ciklo `for` naudojimo pavyzdys:

```
            for (int i = 0; i < 10; i++)
                Console.Write(„{0} „, i * i);
```

Įvykdę ciklą ekrane matysime skaičių nuo 0 iki 9 kvadratus:

```
0 1 4 9 16 25 36 49 64 81
```