5. Programmas izpildes kontrole

Šajā nodaļā apskatīsim C# programmēšanas valodas izteikumu pielietojums, nosacījumu kontroles un atkārtošanas struktūras.

5.1. Izteikumu pielietojums

Izteikums (*expression*) ir operandu (mainīgo) un operatoru (priekšrakstu) kopums. Aplikācijas darbības laikā izteikums tiek novērtēts ar vienu vērtību. Izteikumu garums .NET Framework sistēmā nav ierobežots, tomēr tiek rekomendēts lietot pēc iespējas īsākus izteikumus, jo tas uzlabo koda pārskatāmību un kļūdu meklēšanu nepieciešamības gadījumos.

Izteikumam C# valodā jābeidzas ar `;` simbolu, piemērā ir parādīti vienkāršākie izteikumi:

```
a = a + 1;
k = ( a – b ) / 2;
„Jautājums” + c.ToString();
```

Pēdējā piemēra rindā tiek izmantota divu simbolu virkņu konkatenācijas (apvienošana), kuras realizācijai izmanto `+` simbolu. Šajā piemērā operandi ir `a`, `k`, `b` un `c` mainīgie, savukārt operatori ir `+`, `-`, `/`, `)` un `(` simboli. Operatorus iedala trijās grupās:

- **unārās** – šie operatori izmanto tikai vienu operandu, piemēram, `-` var izmantot kā unāro operatoru negatīvo skaitļu lietošanai;
- **binārās** – izmanto divus operandus; šie operatori ir visplašāk lietojamie;
- **terciārās** – C# valodā ir tikai viens tāds operators `?:`, kuram ir nepieciešami trīs operandi un kurš tiek lietots vienkāršos nosacījumos.

Tabula 5.1.

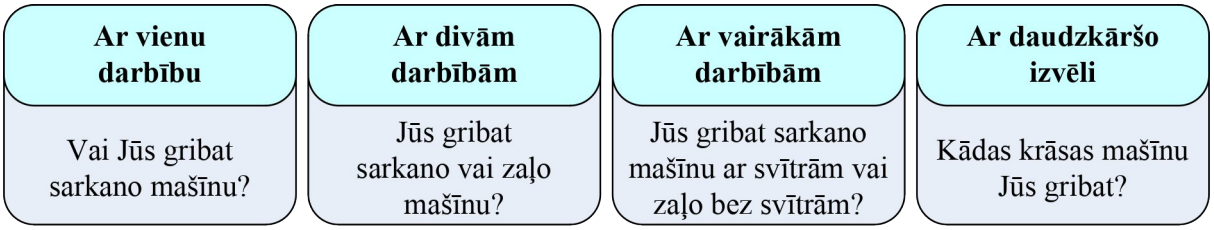
Biežāk lietojamie C# operatori

Operatora veids	Operatori
Aritmētiskie	`+`, `-`, `*`, `/`, `%`
Pieauguma un samazinājuma	`++`, `--`
Salīdzināšanas	`<`, `>`, `<=`, `>=`, `!=`, `==`
Loģiskie	`&&`, `||`, `!`
Indeksācijas	`[]`
Piešķiršanas	`=`, `+=`, `-=`, `*=`, `/=`

Operatori ne vienmēr tiek apstrādāti no kreisās daļas pēc kārtas. Ja **operatoru prioritāte** ir vienāda, tad sākumā tiks izpildīts tas, kas ir no kreisās puses; citos gadījums tiks izpildīts operators ar augstāko prioritāti. Operatoru prioritātes mainīšanai izmanto papildus apaļas iekavas (līdzīgi matemātikai).

5.2. Nosacījumu kontroles struktūras

Nosacījuma kontroles operatori ļauj definēt programmas koda reakciju uz lietotāja darbībām atkarībā no dažādiem nosacījumiem. Pastāv dažādie nosacījuma kontroles operatori: ar vienu darbību, ar divām darbībām, ar vairākām darbībām un ar daudzkāršo izvēli (skat. 5.1.att.).



5.1.att. Nosacījumu kontroles struktūru veidi un piemēri

Nosacījumu ar vienu darbību C# programmēšanas valodā vispārīgā veidā pieraksta šādi:

```
if (nosacījums)
    darbība;
```

Sākumā tiek novērtēts nosacījums, ja tās ir spēkā (nosacījuma loģiskā vērtība ir patiesa (*true*), tad tiek izpildīta darbība. Par darbību var būt jebkurš C# koda bloks; ja bloks sastāv no vairāk kā viena priekšraksta, to jādefinē figūriekavās.

Apskatīsim nosacītu piemēru, kad programmā jārealizē šāds nosacījums: ja darba alga ir lielāka par 320, tad nodoklis ir 100:

```
if (DarbaAlga > 320)
    Nodoklis = 100;
```

Nosacījums ar divām darbībām ir šāds:

```
if (nosacījums)
   darbība1;
else
   darbība2;
```

Šeit atkal sākumā tiek novērtēts nosacījums, ja tās ir spēkā (nosacījuma loģiskā vērtība ir patiesa (*true*), tad tiek izpildīta darbība1, pretējā gadījumā tiek izpildīta darbība2.

Ja darba alga ir lielāka par 320, tad nodoklis ir 100, pretējā gadījumā nodoklis ir 70:

```
if (DarbaAlga > 320)
    Nodoklis = 100;
else
    Nodoklis = 70;
```

Nosacījums ar vairākām darbībām tiek realizēts ar **if/else** struktūru kombināciju. Rezultātā viena **if/else** struktūra ir ielikta otrajā:

```
if (nosacījums)
   darbība1;
else if (nosacījums)
   darbība2;
else
   darbība3;
```

Pieņemsim, ka studenta stipendija ir atkarīga no vidējās sesijas atzīmes: ja vidējā atzīme ir mazāka par 4, tad stipendijas nav; ja tā ir mazāka par 6, tad stipendija ir 20 Ls; ja mazāka par 9, tad 40 Ls; visos pārējos gadījumos stipendija ir 70 Ls):

```
if (vid<4)
     MessageBox.Show(„Stipendijas nav”);
else if (vid<6)
     MessageBox.Show(„Stipendija ir 20 Ls”);
else if (vid<9)
     MessageBox.Show(„Stipendija ir 40 Ls”);
else
     MessageBox.Show(„Stipendija ir 70 Ls”);
```

Līdz šim tika runāts par parastiem nosacījumiem, tomēr pastāv gadījumi, kad ir nepieciešams pielietot saliktus nosacījumus. Tādiem nolūkiem C# valodā pastāv speciālās loģiskās operācijas:

loģisko UN C++ pieraksta šādi: (**a<1 && b<5**);
loģisko VAI C++ pieraksta šādi: (**a<1 || b<5**);
loģisko NĒ C++ pieraksta šādi: **!(a==3);**

Nosacījuma struktūra ar daudzkāršo izvēli ļauj definēt dažādas darbības, kuras tiks izpildītas atkarībā no kāda mainīgā vērtības (šajā gadījumā tās ir 1, 2 un tā tālāk). Šajā struktūrā var paredzēt arī darbību pēc noklusējuma, kura tiks izpildīta tad, ja neviens no definētiem nosacījumiem nav spēkā (to realizē ar

default operatora palīdzību). Katras darbības beigās jāparedz izeja no **switch** struktūras (operators **break**):

```
switch(mainīgais)
     {
     case 1: darbība1; break;
     case 2: darbība2; break;
     ...
     case N: darbībaN; break;
     default: darbība pēc noklusējuma; break;
     }
```

Apskatīsim programmas koda daļu un tās blokshēmu, kur atkarībā no ievadītās paroles tiek izvadīts paziņojums konkrētam lietotājam:

```
switch (parole)
    {
    case 12345:
       MessageBox.Show("Labdien, Ivans");
       break;
    case 23456:
       MessageBox.Show("Labdien, Laima");
       break;
    case 34567:
       MessageBox.Show("Labdien, Modris");
       break;
    default:
       MessageBox.Show("Ieeja aizliegta");
       break;
    }
```

Ja parole atbilst kādam no definētiem gadījumiem, programma sveicina lietotāju, visos pārējos gadījumos izvada paziņojumu, ka ieeja ir aizliegta.

5.3. Atkārtošanas struktūras

Atkārtošanas struktūras (cikli) ļauj programmētājam noteikt darbību, kurai jāizpildās līdz brīdim, kamēr izpildās kāds nosacījums. .NET Framework piedāvā četrus ciklu veidus:

1. **for** cikls ļauj atkārtot tajā definēto kodu norādīto reižu skaitu. Šīm nolūkam izmanto speciālo mainīgi, kuru sauc par cikla skaitītāju un kura vērtība mainās cikla izpildes laikā.
2. **foreach** cikls paredzēts kolekciju un masīvu apstrādei. Tā funkcionalitāte ir identiska for ciklam, bet pieraksta veids ir vienkāršāks un īsāks.

3. **while** cikls arī atkārto noteiktas darbības vairākās reizes, tomēr šeit cikla skaitītāja izmantošana nav obligāta.
4. **do/while** cikls no pārējiem atšķiras ar to, ka tajā definētais kods obligāti tiks izpildīts vismaz vienu reizi, neatkarīgi no cikla beigšanas nosacījuma. Šajā gadījumā sākumā tiek izpildīts cikla kods un tikai pēc tam notiek cikla nosacījuma pārbaude.

Vispārīgā veidā **for** cikla struktūra tiek definēta šādi:

```
for (inicializācija; nosacījums; pieaugums)
     darbība;
```

Ar inicializāciju saprot cikla mainīga definēšanu un tās sākotnējās vērtības noteikšanu. Tālāk tiek pārbaudīts cikla izpildes nosacījums, bet pieaugums definē cikla skaitītāja pieauguma vai samazinājuma soli. Apskatīsim cikla piemēru, kurš atkārto noteikto kodu (darbības) piecas reizes:

```
for (int i=1; i<=5; i++)
     {
     darbības;
     }
```

foreach cikla struktūra ir šāda:

```
foreach (mainīgais in kolekcija)
     darbība;
```

Šeit mainīgais ir paredzēts iterāciju atkārtošanai, pēc kura pieraksta kolekcijas nosaukumu, kuras elementus ir nepieciešams apstrādāt ciklā. Apskatīsim šī cikla izmantošanas piemēru:

```
foreach (int j in mas)
    {
    string t = mas[j].ToString();
    textBox1.Text = textBox1.Text+t;
    }
```

Šajā piemērā ar **foreach** cikla palīdzību tiek apstrādāts masīvs ar nosaukumu **mas**. Cikla pārskata visas masīva elementus un izvada tās vērtība uz formas teksta laukumā. Galvenā šī cikla priekšrocība ir tas, ka programmētājam nav jāatceras masīva elementu skaits un cikla kods ir ļoti kompakts.

Vispārīgā veidā **while** ciklu pieraksta šādi:

```
//cikla skaitītāja inicializācija;
while (nosacījums)
   {
   darbība vai darbības;
```

```
    //skaitītāja pieaugums;
    }
```

Šeit cikla skaitītāja inicializācija un tās pieaugums aizkomentēti tādēļ, ka šim ciklam tas nav obligāti, daudzos gadījumos pietiek ar to, ka norāda tikai cikla beigas nosacījumu.

Pēdējais cikla paveids ir do/while, kura vispārīgais pieraksta veids ir:

```
do
    {
    darbība vai darbības;
    }
while (nosacījums)
```

Ciklos ir iespējams lietot operatorus **break** un **continue**. Kad operators **break** tiek izpildīts ciklos, notiek tūlītēja izeja no cikla, bet programma turpina izpildīt nākošus operatorus pēc cikla. Parasti operatoru **break** izmanto cikla pārtraukšanai, bet operators **continue** izlaiž operatorus, kuri iet uzreiz pēc tā, un tiek izpildīts nākošā cikla iterācija.

Pastāv vispārīgie ieteikumi dažādu ciklu veidu pielietošanai:

- izmantojiet for ciklu tad, ja gribat atkārtot kodu noteiktu reižu skaitu;
- izmantojiet for ciklu gadījumos, kad ir nepieciešams izmantot cikla skaitītāja vērtību masīva indeksiem un tam līdzīgiem gadījumiem;
- izmantojiet foreach ciklu kolekciju un masīvu vienkāršotai apstrādei;
- izmantojiet ciklu while koda atkārtošanai nulle vai vairāk reižu;
- izmantojiet ciklu do/while ja gribat izpildīt kodu vismaz vienu reizi.

Nodaļas kopsavilkums

Izteikums ir operandu (mainīgo) un operatoru (priekšrakstu) kopums. Aplikācijas darbības laikā izteikums tiek novērtēts ar vienu vērtību. Izteikumu garums .NET Framework sistēmā nav ierobežots, tomēr tiek rekomendēts lietot pēc iespējas īsākus izteikumus, jo tas uzlabo koda pārskatāmību un kļūdu meklēšanu nepieciešamības gadījumos.

Nosacījuma kontroles operatori ļauj definēt programmas koda reakciju uz lietotāja darbībām atkarībā no dažādiem nosacījumiem. Pastāv dažādie nosacījuma kontroles operatori: ar vienu darbību (**if**), ar divām darbībām(**if**…**else**), ar vairākām darbībām (**if** … **else if** …**else**) un ar daudzkāršo izvēli (**switch**).

Atkārtošanas struktūras (cikli) ļauj programmētājam noteikt darbību, kurai jāizpildās līdz brīdim, kamēr izpildās kāds nosacījums. .NET Framework piedāvā četrus ciklu veidus: **for**, **foreach**, **while** un **do/while**.

UZDEVUMI PAŠPĀRBAUDEI

1. Uzrakstīt programmu, kura atkarībā no ievadītas atzīmes izdod paziņojumu: **Ieskaitīs** – ja atzīme ir [4; 10]; **Nav ieskaitīts** – ja atzīme ir [0; 3]; **Tāda atzīme nevar būt** – ja atzīme ir ārpus iepriekš dotiem intervāliem.
2. Uzrakstīt programmu, kura ļauj ievadīt divus skaitļus, tad salīdzina šos skaitļus un izdod šādu informāciju par visām iespējamām attiecībām starp ievadītiem skaitļiem.
3. Uzrakstīt programmu, kura aprēķina šādu vienādojumu sistēmu:

$$y = \begin{cases} (a + b)^2 - c & , \text{ja } c<0; \\ (3a - b)^3 + 2 + c & , \text{ja } c>0; \\ 3(a + b + c) & , \text{ja } c=0. \end{cases}$$

4. Uzrakstīt programmu, kura aprēķina skaitļu virkni:
   $y_1 = x * (x - 1)$; $y_2 = (x - 1)*(x - 2)$ un tā tālāk līdz $y_n = (x - n - 1) * (x - n)$. Programma ļauj ievadīt skaitļus x un n un attēlo uz formas visus aprēķinu rezultātus.
5. Uzrakstīt programmu, kura izveido viendimensiju 20 elementu garu masīvu, kur katrs nākošais masīva elements ir lielāks par iepriekšējo par 5. Pirmo masīva elementu ievada lietotājs. Tad izvadīt visus masīva elementus uz ekrāna ar foreach ciklu textBox objektā.

UZDEVUMI PATSTĀVĪGAJĀM DARBAM

1. Uzrakstīt programmu, kura izvada pār- vai nepārskaitļus pēc lietotāja izvēles ar norādīto skaitļu skaitu.
2. Ar **`for`** cikla palīdzību uzrakstīt programmu, kura aprēķina skaitļa faktoriālu (n! = 1 * 2 * 3 * … * n). Ja skaitlis ir <=0, tad paziņot, ka skaitlis nav pareizs.
3. Uzrakstīt programmu, kura izveido viendimensiju 5 elementu garu masīvu, kur lietotājs pats var noteikt masīva elementu vērtības. Tad programma aprēķinā masīva elementu summu un izvada šo rezultātu uz ekrāna paziņojumu loga veidā (ar MessageBox objektu).