Veiksmų šakojimas

Paprasčiausio sąlyginio sakinio taikymas

Gyvenime labai dažnai pasitaiko situacijų, kai reikia pasirinkti, ar eiti tiesiai, ar pasukti į šalį. Kartais tenka rinktis ir iš didesnio kiekio variantų skaičiaus, kadangi programuodami sprendžiame gyvenimiškas problemas, tai ir programavimo kalboje yra priemonių, leidžiančių pasirinkti vieną veiksmą iš dviejų. Toks veiksmų parinkimas vadinamas **Šakojimu**, o jam naudojamas **sąlygos sakinys**.

Sutrumpintas sąlyginis sakinys naudojamas paprasčiausiu atveju, kai reikia atlikti vieną sakinį, jei tenkinama tam tikra sąlyga:	`if (sąlyga) Sakinys;`

Jei sąlyga tenkinama, atliekamas pirmasis sakinys, jei netenkinama - antrasis:	`if (sąlyga) PirmasisSakinys;` `else AntrasisSakinys;`

1 pavyzdys. Nurodyti du nelygūs sveikieji skaičiai `a` ir `b`.

Iš didesniojo skaičiaus reikia atimti mažesnįjį:	`if (a > b) a = a - b;` `else b = b - a;`
Tai galima užrašyti ir trumpiau:	`if (a > b) a -= b;` `else b -= a;`

2 pavyzdys. Jei skaičius `sk` yra neigiamas, reikia jam pakeisti jo ženklą:

Jei skaičius `sk` yra neigiamas, reikia jam pakeisti jo ženklą:	`if (sk < 0) sk = - sk;`

Jei sąlyga netenkinama, tai sakinys esantis po sąlygos, neatliekamas ir pereinama prie tolesnių veiksmų.

Daugiau nei du pasirinkimo atvejai

Ką daryti, jei yra daugiau nei du pasirinkimo atvejai? Išeitis labai paprasta: vieną sąlygos sakinį reikia įterpti į kitą sąlygos sakinį.

Pavyzdys. Tarkime turime du sveikuosius skaičius. Reikia sužinoti, ar jie yra lygūs, ar kuris nors yra didesnis. Išskiriame tris atvejus:

```
if (a == b) cout << "Skaičiai yra lygūs" << endl;
else if (a > b) cout << "Skaičius a yra didesnis" << endl;
   else cout << "Skaičius b yra didesnis" << endl;
```

Tam tikras sakinių lygiavimas pagerina programos vaizdumą, ją galima lengviau skaityti. Pavyzdyje sakinys, kuris vykdomas, jei sąlyga tenkinama, parašytas iš karto po sąlygos. Kartais taip patogiau ir skaičiuoti.

Loginiai reiškiniai

Išnagrinėkime kelis pavyzdžius.

1 pavyzdys. Skaičius `a` turi būti teigiamas. Šią sąlygą (loginį reiškinį) rašome taip:

```
a > 0
```

2 pavyzdys. Tarkime skaičius `a` turi būti dviženklis. Paparasčiausia bus pareikalauti, kad jis būtų didesnis už 9 ir mažesnis už 100. Sąlygoje atsirado žodelis **ir**, kuri reikia užrašyti C++ kalbos žymėmis. Tai atrodo taip:

```
a > 9 && a < 100
```

Kasdieninėje kalboje vartojami žodžiai **ir, arba** ir **ne** atitinka pagrindinius loginius nveiksmus, kurie programavime vadinami **loginiais operatoriais**.

3 pavyzdys. Užrašykime tokią sąlygą: skaičius `a` turi būti lyginis arba turi nesidalyti iš 7:

```
a % 2 = 0 || a % 7 != 0
```

Loginiai veiksmai

Loginis veiksmas	Operatorius	Prasmė	Pavyzdys	Paaiškinimas
ne	**!**	Neigimas arba inversija	!(a > 3)	Reikšmę true keičia false ir atvirkščiai font false keičia į true.
ir	**&&**	Loginė daugyba	(x > 2 && x < 5)	Įgyja reikšmę true, kai visų loginių reikšmių, sujungtų ženklais &&, reikšmė yra true.
arba	**||**	Loginė sudėtis	a < 0 || a > 3	Įgyja reikšmę false, kai visų loginių reikšmių, sujungtų ženklais &&, reikšmė yra false.

Skaičių palyginimas

Skaičiams palyginti C++ kalboje naudojami šeši operatoriai, tik dėl riboto klaviatūros simbolių skaičiaus juos tenka šiek tiek kitaip užrašyti, nei priimta matematikoje.

C++ kalbos operatoriai	Matematikos ženklai
>	>
<	<
==	=
!=	≠
>=	≥
<=	≤

Veiksmų atliko tvarka aritmetiniuose ir loginiuose reiškiniuose

Eiliškumas	Operatoriai						Pavyzdžiai	
1	()						4 - (x + 5 * y)	
2	-	!					-a + 3 * b	!(x >5)
3	*	/	%				a * b / 3	sk % 10
4	+	-					x + y - 3	

5	>	<			>=	<=	x > 5 - y)	a <= 8
6	==	!=					a == b	((a != c)
7	&&						a > b && b == c	
8	||						a > b || a > c	
9	=	+=	-=	*=	/=	%=	b *= 5;	a = b + c;

Apskaičiuojant **loginių reiškinių** reikšmes, pirmiausia ieškoma ženklo keitimo arba neigimo veiksmų (operatorių), paskui - daugybos ir dalybos veiksmų, tada - sudėties ir atimties veiksmų ir t.t.

Jei reikia, veiksmų tvarką galima pakeisti skliaustais. Pavyzdžiui, reiškinys $y > \frac{b}{3-x}$ užrašomas eilute naudojant sakinius:

```
y > b / (3 - x);
```

Kadangi taip užrašytas sudėtingas loginis reiškinys atrodo nevaizdžiai, jį sunku suprasti, tai dažnai specialiai paliekami skliaustai, kurie veiksmų tvarkos net nekeičia. Anksčiau pateiktą 2 pavyzdį galima užrašyti ir taip:

```
(a > 9) && (a < 100)
```

o 3 pavyzdį - taip:

```
(a % 2 = 0) || (a %7 != 0)
```

4 pavyzdys. Nurodyti skaičiai a, b, ir c, kurie reiškia atkarpų ilgius. Sudarykime loginį reiškinšį, kurio reikšmė būtų `true`, jei iš tų atkarpų galima sudaryti trikampį.

Tam pasinaudosime trikampio nelygybe: kai bet kurių dviejų kraštinių ilgių suma didesnė už trečios kraštinės ilgį, trikampį sudaryti galima. Atsižvelgdami į veiksmų eiliškumą, nesunkiai aprašysime sąlygą:

```
a + b > c && a + c > b && b + c > a
```

Pirmiausia atliekami sudėtiniai veiksmai, paskui - palyginimo, galiausiai - loginiai. Programa, kuri sprendžia šį uždavinį gali atrodyti taip:

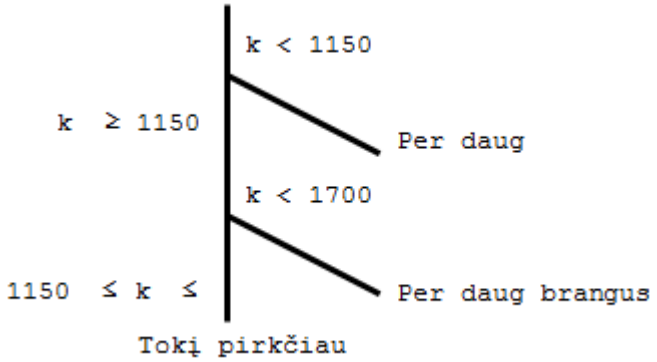
```
int main()
{ int a, b, c;
cout << "Kokie krastiniu a, b ir c ilgiai? ";
cin >> a >> b >> c;
if (a + b > c && a + c > b && b + c > a)
  cout << "trikampi sudaryti galima" << endl;
  else cout << "trikampi sudaryti negalima" << endl;
return 0;
}
```

Sudėtingesnės sąlygos

1 pavyzdys. Jurgis rengiasi pirkti kompiuterį. Apie kompiuterio kainą jis samprotauja taip:

- jei kompiuteris pigesnis nei 1150 €, tai jis per pigus;
- jei kompiuteris brangesnis nei 1700 €, tai jis per daug brangus;
- jei kompiuteris kainuoja nuo 1150 € iki 1700 €, tai jis yra toks, kokį pirkčiau.

Šiuos samprotavimus galima pavaizduoti grafiškai:

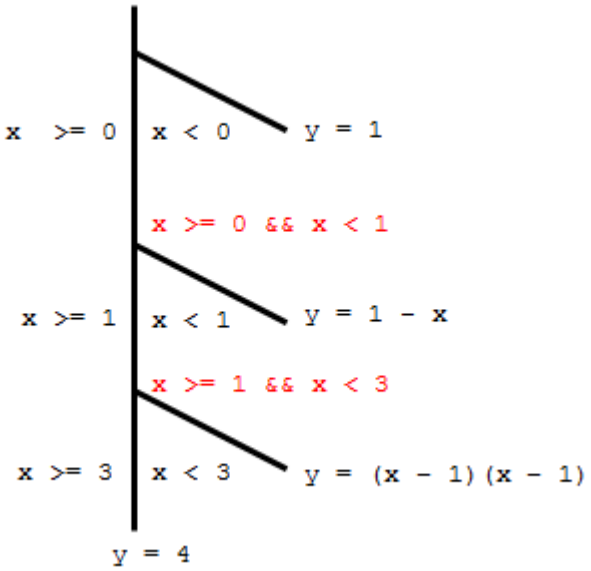


Patikriname pirmus du atvejus, trečiojo atvejo tikrinti nereikia - jis bus tenkinamas savaime.

Sudarykime programą:

```
#include < iostream >
using namespace std;
int main ()
{
int k;
cin >> k;
if (k < 1150) cout << "Per daug pigus" << endl;
else if (k > 1700) cout << "Per daug brangus" << endl;
  else cout << "Toki pirkciau" << endl;
return 0;
}
```

2 pavyzdys. Kuri apskaičiuotų funkcijos $y = \begin{cases} 1, kai\ x < 0, \\ 1 - x, kai\ 0 \le x < 1, \\ (x-1)^2, kai\ 1 \le x < 3, \\ 4, kai\ x \ge 3 \end{cases}$ reikšmes.



Skaičiavimus pavaizdavę grafiškai, matome, kad galime sutaupyti loginių sąlygų tikrinimo veiksmų. Kartais tuo įmanoma pasinaudoti.Programa atrodytų taip:

```
int main ()
{ double x, y;
cin >> x;
if (x < 0) y = 1;
else if (x < 1) y = 1 - x;
 else if (x < 3) y = (x - 1) *(x - 1);
  else y = 4;
cout << y << endl;
return 0;
}
```

Galima rašyti ir tiksliai taip, kaip pateikta sąlygoje. Dėl to programa nepasidarys paprastesnė, bet taps suprantamesnė:

```
int main ()
{ double x, y;
cin >> x;
if (x < 0) y = 1;
else if (x < 1) && (x < 1)) y = 1 - x;
 else if ((x >= 1) && (x < 3)) y = (x - 1) *(x - 1);
  else y = 4;
cout << y << endl;
return 0;
}
```

Užtenka patikrinti tik vieną sąlygos pusę, nes kita jau būna patikrinta atliekant prieš tai buvusį sąlyginį sakinį. Jei sąlyga x < 0 netenkinama, vadinasi, tenkinama x >= 0. Toliau vykdomas sakinys `else`, kuriame yra sąlyga x < 1, taigi savaime tikrinama sąlyga sakinyje yra tik x < 1.