1 pamoka

Pirmas žvilgsnis į C++

Šioje pamokoje išnagrinėsime operatorius, iš kurių susideda C++ programa. Pamatysite, kad C++ programose laikomasi to paties formato: pradedama nuo vieno ar keleto operatorių *#include*, yra eilutė *void main(void)*, po to seka operatorių rinkiniai, sugrupuoti tarp kairiojo ir dešiniojo figūrinių skliaustų. Iš šios pamokos suprasite, kad šiuos operatorius suprasti labai paprasta. Šios pamokos gale jūs susipažinsite su tokiomis koncepcijomis kaip:

- Operatorius *#include* suteikia išorinių failų panaudojimo pirmenybę, kuriuose yra C++ operatoriai ar programos apibūdinimai;
- Pagrindinė C++ programos dalis prasideda operatoriumi *void main(void)*;
- Programa susideda iš vienos ar kelių funkcijų, kurios, susideda iš operatorių, skirtų spręsti tam tikrai užduočiai;
- Jūsų programose teksto išvedimui bus plačiai naudojamas išėjimo srautas *cout*.

Kai kuriate C++ programą, realiai jūs dirbate su operatoriais, bet ne su instrukcijomis. Vėliau jūs išstudijuosite priskyrimo operatorius, kurie priskiria kintamiesiems reikšmes, operatorių *if* , kuris leidžia programai priiminėti sprendimus.

Panagrinėkime paprasčiausią programą:

```
#include <iostream.h>
void main(void)
 {
      cout <<”mokomės programuoti C++ kalba!”;
 }
```

Joje yra trys operatoriai. Figūriniai skliaustai (vadinami grupuojančiais) grupuoja susietus operatorius:

```
#include <iostream.h>
void main(void)
 {
      cout <<”Mokomės programuoti su C++ kalba!”;
 }
```

Programos operatorius

Dažnai programa prasideda operatoriumi *#include*. Pavyzdžiui:

```
#include <iostream.h>
```

Kompiliuojant programą, operatorius *#include* priverčia kompiliatorių prijungti išorinio failo *iostream.h* turinį prie jūsų programos. Failai su plėtiniu *.h*, kuriuos prijungiate pradžioje jūsų programos, vadinami tiesiog "h" failais. Jei pažvelgsite į kompiliatoriaus failų katalogus, rasite katalogą INCLUDE, kuriame yra įvairūs išoriniai failai. Kiekvienas išorinis failas turi aprašymą, kuris pateikiamas kompiliatoriui skirtingoms operacijoms. Pavyzdžiui, egzistuoja tokie išoriniai failai, kurie turi aprašymus matematinėms operacijoms, kitas išorinis failas aprašo failines operacijas ir t.t..

Išoriniai failai vaizduojami ASCII formate, todėl jų turinį galite matyti ekrane ar atspausdinti . Kad geriau suprastumėte išorinių failų turinį, atsispausdinkite failą IOSTREAM.H, kurio turinį naudosite kiekvienoje jūsų kuriamoje C++ programoje. Paprastai išorinis failas IOSTREAM.H yra INCLUDE pakatalogyje, kuris yra kataloge, kuriame saugomi C++ kompiliatoriaus failai. Kad pažiūrėti ar atsispausdinti išorinį failą, naudokite tekstinį redaktorių.

Pastaba: niekada nekeiskite išorinių failų turinio, nes kompiliuojant jūsų programą gali atsirasti klaidų.

Kas yra void main(void)

Kuriant C++ programas jūsų išeities faile bus daugelis operatorių. Mokydamiesi suprasite, kad tvarka, kuria operatoriai atsiranda programoje, nebūtinai turi sutapti su tvarka, kuria operatoriai bus

vykdomi paleidžiant programą. Kiekviena C++ programa turi vieną įėjimą, nuo kurio pasideda programos vykdymas, t.y. pagrindinę programą. C++ programoje operatorius *void main(void)* parodo jūsų programos pradžią.

Kai programos tampa didesnės ir sudėtingesnės, jos dalinamos į kelias mažesnes. Šiuo atveju operatorius *void main(void)* nurodo pradinius (arba pagrindinius) programos operatorius – programos dalis, kuri vykdoma pirmiausia. Programoje turi būti vienas ir tik vienas operatorius vardu *main*. Peržiūrint dideles C++ programas ieškote *main*, kad nustatyti operatorius, nuo kurių prasideda programos vykdymas.

void naudojimas

Kai tik jūsų programa tampa sudėtinga, tikslinga ją suskaidyti į dalis, vadinamas funkcijomis. Funkcija susideda iš paprasto operatorių rinkinio, kurie atlieka tam tikra užduotį. Pavyzdžiui, kuriant mokesčių dokumentų programą, jūs galite sukurti funkciją *salary*, skaičiuojančią tarnautojų darbo užmokestį. Analogiškai jei rašote matematinę programą, galite sukurti funkcijas su vardais *square_root* arba *cube*, kurios pateikia tam tikrų matematinių veiksmų rezultatus. Jei programa naudoja funkciją, pastaroji , atlikusi savo užduotį, rezultatą grąžina pagrindinei programai.

Programoje funkcija turi gauti unikalų vardą.

Žodis *void* nurodo, kad funkcija negrąžina jokio rezultato arba jokios reikšmės jai neperduodamos. Pavyzdžiui, jei jūs naudojate MS – DOS arba UNIX terpę, programa gali baigti savo vykdymą grąžindama ką nors operacijų sistemai. Tai gali būti patikrinama komandiniame faile. MS – DOS komandiniai failai tikrina programos išėjimo būseną naudojant komandą IF ERRORLEVEL. Pavyzdžiui, tarkime, kad PAYROLL.EXE baigiasi viena iš duotų būsenos reikšmių, priklausomai nuo atlikimo rezultatų:

Būsena	Reikšmė
0	Sėkmingai
1	Failas nerastas
2	Spausdintuve nėra popieriaus

MS – DOS komandinio failo viduje jūs galite patikrinti programos darbo rezultatą naudojant komandą IF ERRORLEVEL:

```
PAYROLL
IF  ERRORLEVEL   0  IF NOT ERRORLEVEL 1 GOTO SUCCESSFUL
IF  ERRORLEVEL   1  IF NOT ERRORLEVEL 2 GOTO NO_FILE
IF  ERRORLEVEL   2  IF NOT ERRORLEVEL 3 GOTO NO_PAPER
REM Toliau eina kitos komandos
```

Dauguma C++ programų, kurios bus sukurtos nagrinėjant šią knygą, nepateikia jokių būsenos reikšmių operacijų sistemai. Šiuo atveju naudoti žodį *void* prieš *main*, kaip parodyta žemiau:

```
void main(void)
```

programa nepateikia reikšmių

C programa gali naudoti informaciją (pvz., failo pavadinimą), kurią vartotojas nurodo komandinėje eilutėje paleisdamas programą. Jei programa nenaudoja komandinės eilutės informacijos, jūs turite patalpinti žodį *void* tarp skliaustų po žodžio *main*, kaip parodyta žemiau:

```
void main(void)
```

programa nenaudoja komandinės eilutės argumento

Grupavimo operatorius {}

Programose gali būti operatorių rinkinys, kurį kompiuteris privalo įvykdyti kelis kartus arba kai yra patenkinta tam tikra sąlyga. Kad sugrupuoti susietus operatorius, C++ programose naudojami figūriniai skliaustai {}.

cuot panaudojimas teksto išvedimui į ekraną

Teksto išvedimui galima panaudoti *cout* ir dvigubą ženklą "mažiau" (<<), kaip parodyta žemiau:

```
cout <<"labas, C++";
```

Žodis *cout* reiškia išeinantį srautą, kuris nukreiptas į standartinį operacinės sistemos išvedimo įrenginį. Pagal nutylėjimą operacinė sistema kaip standartinį išvedimo įrenginį nurodo displėjaus ekraną. Kad išvesti pranešimą į ekraną, jūs paprasčiausiai naudojate dvigubą simbolį "mažiau" (tai įterpimo operatorius) su išeinančiu srautu *cout.*

Įterpimo operatorius (<<) leidžia programai įterpti simbolius į išeinantį srautą. Panaudojus operacinės sistemos peradresavimo operatorius, programos pranešimą galima pasiųsti į spausdintuvą ar failą. Pavyzdžiui, ši komanda nurodo MS – DOS nukreipti programos FIRST.EXE išėjimą į spausdintuvą, o ne į ekraną:

```
C:\>FIRST>PRN<ENTER>
```

Ką turite žinoti?

Šioje pamokoje buvo aptarti keli bendri klausimai, su kuriais susidursite kuriant C++ programas. Prieš pradedant studijuoti kitą pamoką, įsitikinkite, kad susipažinote su šiomis pagrindinėmis koncepcijomis:

- ☑ Dauguma C++ programų prasideda operatoriumi *#include*, kuris nurodo kompiliatoriui prijungti nurodyto išorinio failo turinį.
- ☑ Išoriniuose failuose laikomi nurodymai kompiliatoriui, kuriuos jūsų programa gali naudoti.
- ☑ Išeities failas gali būti sudarytas iš daugelio operatorių; operatorius *void main(void)* nurodo pagrindinės programos pradžią, kurioje yra pirmas vykdomas programos operatorius.
- ☑ Kai tik jūsų programa tampa labiau sudėtinga, jums reikia suskaidyti ją į nedideles lengvai valdomas dalis,vadinamas funkcijomis. Programos operatoriai grupuojami su figūriniais skliaustais {}.
- ☑ Dauguma C++ programų informacijos išvedimui į ekraną naudoja išėjimo srautą *cout*; tačiau naudojant operacinės sistemos įėjimo/išėjimo peradresavimą galima *cout* nukreipti į failą, įrenginį (pavyzdžiui, spausdintuvą), arba jį padaryti kitos programos įėjimu.

2 pamoka

Pranešimų išvedimas

Šioje pamokoje *cout* panaudosime simbolių, sveikų skaičių ir skaičių su slankiu kableliu išvedimui. Šios pamokos pabaigoje jūs būsite susipažinę su šiomis koncepcijomis:

- Simbolių ir skaičių išvedimui į ekraną galite naudoti išeinantį srautą *cout*.
- C++ kalboje su *cout* galima naudoti specialius simbolius (tabuliacijos) naujos eilutės išvedimui, o taip pat garso išvedimui jūsų kompiuteryje.
- C++ kalboje skaičius galima pavaizduoti dešimtainiu, aštuntainiu ar šešioliktainiu formatu.
- Su operacinės sistemos komandinės eilutės peradresavimo operatoriumi galima savo programos išeinančius srautus nukreipti į failą ar spausdintuvą.
- Naudojant išėjimo srautą *cerr*, jūsų programos pranešimus gali siųsti į standartinį klaidų įrenginį.
- Jūs galite formuoti išvedimą naudojantis srauto viduje esančiu modifikatoriumi *setw*.

cout panaudojimas skaičių išvedimui

Iki šiol sukurtos programos naudojo *cout* simbolių eilučių išvedimui (raidžių ar skaičių, paimtų į kabutes). *cout* galima panaudoti skaičių išvedimui. Sekanti programa 1001.CPP į ekraną išveda skaičių 1001:

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

Sukompiliuokite ir paleiskite šią programą. Ekrane bus atvaizduotas skaičius 1001, kaip parodyta žemiau:

```
C:\1001 <ENTER>
1001
```

Toliau redaguokite programą ir pakeiskite operatorių *cout* taip, kad būtų išvestas skaičius 2002, kaip parodyta žemiau:

```
cout <<2002;
```

Be sveikų skaičių vaizdavimo (skaičių be dešimtainio kablelio), *cout* leidžia vaizduoti skaičius su slankiu kableliu, pavyzdžiui 1.2345. Programa FLOATING.CPP *cout* naudoja skaičiaus 0.12345 išvedimui į ekraną:

```
  #include <iostream.h>
void main(void)
  {
   cout <<0.12345;
  }
```

Kaip ir anksčiau, sukompiliuokite ir paleiskite programą. Ekrane pasirodys toks vaizdas;

```
C:\> FLOATING <ENTER>
0.12345
```

Keleto reikšmių išvedimas vienu metu

Kaip jau žinote, dvigubas ženklas “mažiau” yra įterpimas (ši operacija įterpia simbolius į išeinantį srautą). *cout* pagalba galima naudoti keletą įterpimo operacijų vieno operatoriaus ribose. Pavyzdžiui, programa 1001TOO.CPP naudoja šią operaciją skaičiaus 1001 atvaizdavimui ekrane keturis kartus:

```
#include <iostream.h>
void main(void)
  {
   cout <<1 <<0 <<0 <<1;
  }
```

Kai sukompiliuosite ir paleisite šią programą, ekrane pasirodys štai kas:

```
C:\> 1001TOO <ENTER>
1001
```

Kiekvieną kartą, kai sutinkama įterpimo operacija, skaičiai arba simboliai pridedami prie tų , kurie jau yra išeinančiame sraute. Programa SHOW1001.CPP *cout* pagalba išveda simbolių eilutę ir skaičių:

```
#include <iostream.h>
void main(void)
  {
   cout << "Mano mėgstamiausias skaičius yra " <<1001;
  }
```

Atkreipkite dėmesį, kad tarpas, sekantis po žodžio *yra* (kabučių viduje), skirtas skaičiaus 1001 atskyrimui. Be tarpo skaičius susilietų su žodžiu (yra1001). Panašiu būdu programa 1001MID.CPP vaizduoja skaičių 1001 simbolinės eilutės viduryje:

```
#include <iostream.h>
void main(void)
  {
   cout << "Skaičius " <<1001 << " man labai patinka";
  }
```

Programa MIXMATCH.CPP kombinuoja eilutes, simbolius, sveikus skaičius ir skaičius su plaukiojančiu kableliu to paties išeinančio srauto viduje:

```
#include <iostream.h>
void main(void)
  {
   cout << "Kai man buvo " <<20 << " metų, mano indėlis buvo "
          << 493.34 << end1;
  }
```

Kai sukompiliuosite ir paleisite šią programą, ekrane pasirodys toks pranešimas:

```
C:\> MIXMATCH <ENTER>
Kai man buvo 20 metų, mano indėlis buvo 493.34
```

Specialių simbolių panaudojimas išvedimui programuoti

Visos iki šiol jūsų sukurtos programos išvedamą pranešimą vaizdavo viena eilute. Tačiau programos, kurias jūs kursite ateityje, galėtų atvaizduoti kelias pranešimo eilutes. Tarkime, kad jūs kuriate programą, kuri ekrane išves adresus. Tikriausiai jūs norėsite, kad adresai atsirastų kelių eilučių pavidalu.

Tam, kad perkelti kursorių į kitos eilutės pradžią, galima įterpti *naujos eilutės simbolį* (\n) į išeinantį srautą. C++ kalboje galimi du naujos eilutės formavimo būdai. Pirma, galima patalpinti \n simbolį išeinančio srauto viduje. Programa TWOLINES.CPP vaizdą pateikia dviejų eilučių pavidalu naudojant naujos eilutės simbolį:

```
#include <iostream.h>
void main(void)
  {
   cout << "Ši eilutė pirma\nŠi eilutė antra";
  }
```

Kai sukompiliuosite ir paleisite šią programą, naujos eilutės simbolis leis pateikti vaizdą dviejų eilučių pavidalu, kaip parodyta žemiau:

```
C:\> TWOLINES <ENTER>
Ši eilutė pirma
Ši eilutė antra
```

Programa NEWLINES.CPP išveda skaičius 1,0,0,1 skirtingose eilutėse:

```
#include <iostream.h>
```

```
void main(void)
  {
   cout << 1 << '\n' <<0 << '\n' <<0<< '\n' <<1;
  }
```

Kursoriaus perstūmimui į sekančią eilutę galima panaudoti simbolį *endl* (eilutės pabaiga). Programa ENDL.CPP iliustruoja *endl* panaudojimą kursoriaus perstūmimui į naujos eilutės pradžią:

```
#include <iostream.h>
void main(void)
  {
   cout << "O dabar…" <<endl
          << "Mokomės programuoti C++ kalba";
  }
```

Kaip ir anksčiau, kai sukompiliuosite ir paleisite programą, ekrane vaizdas bus pateiktas dviejų eilučių pavidalu.

C:\ > ENDL <ENTER>
O dabar
Mokomės programuoti C++ kalba

Programa ADDRESS.CPP išves fakulteto adresą keliose eilutėse:

```
#include <iostream.h>

void main(void)
 {
    cout << "KTU"<< end1;
    cout << "Telekomunikaciju ir elektronikos fakultetas" << end1;
    cout << "LT 3031 Studentu 50" << end1;
  }
```

Šalia naujos eilutės simbolio, leidžiančio perkelti kursorių į naujos eilutės pradžią, galite panaudoti specialius simbolius, kurie išvardinti lentelėje.

Kursoriaus valdymo simboliai

Simbolis	Paskirtis
\a	Signalinis (skambučio) simbolis
\b	Sugrąžinimo simbolis
\f	
\n	Naujos eilutės simbolis
\r	
\t	Horizontalios tabuliacijos simbolis
\v	Vertikalios tabuliacijos simbolis
\\	
\?	Klausimo ženklas
\'	Vienos kabutės
\''	Dvigubos kabutės
\0	Nulinis simbolis
\ooo	Aštuonetainė reikšmė, pvz.\007
\xhhh	Šešioliktainė reikšmė,pvz\xFFFF

Pastaba: naudojant valdymo simbolius reikia juos talpinti viengubose kabutėse, jei juos vartojate atskirai, pvz. '\n', arba dvigubose kabutėse, jeigu naudojate juos eilutės viduryje, pvz. "Sveiki\nTai As!".

Programa SPECIAL.CPP naudoja specialius signalo (\a) ir tabuliacijos (\t) simbolius garso generavimui ir žodžių išvedimui:

```
#include <iostream.h>

void main(void)
  {
     cout << “Skambutis\a\tSkambutis\a\tSkambutis\a”;
   }
```

Aštuonetainių ir šešioliktainių skaičių išvedimas

Skaičių išvedimui aštuonetainiame ar šešioliktainiame formatuose galima panaudoti modifikatorius dec, oct ir hex išvedamo srauto viduje. Programa OCTNEX.CPP juos panaudoja skaičių išvedimui dešimtainiame, aštuonetainiame ir šešioliktainiame pavidaluose:

```
#include <iostream.h>

void main(void)
  {
     cout << “Aštuonetainis: “ << oct << 10 << ‘ ‘ << 20 << end1;
     cout << “Šešioliktainis: “ << hex << 10 << ‘ ‘ << 20 << end1;
     cout << “Dešimtainis: “ << dec << 10<< ‘ ‘ << 20 << end1;
   }
```

Kai sukompiliuosite ir paleisite šią programą, ekrane pasirodys sekantis rezultatas:

```
C:\> OCTNEX <ENTER>
Aštuonetainis: 12  24
Šešioliktainis:  a  14
Dešimtainis: 10  20
```

Pastaba: pasirinkus vieną iš modifikatorių, jis veiks iki tol, kol pasibaigs programa arba kol jo nepakeisite kitu.

Išvedimas į standartinį klaidų įrenginį

Kaip jūs jau žinote, naudojant cout, galima peradresuoti programos išvedimą į įrenginį ar failą operacinės sistemos perskirstymo operatorių pagalba. Jei jūsų programai riekia išvesti pranešimą apie klaidą, jūs turite naudoti išėjimo srautą *cerr*. C++ sujungia *cerr* su standartiniu operacinės sistemos klaidų įrenginiu. Programa CERR.CPP naudoja išėjimo srautą *cerr* pranešimo “Programos klaida” išvedimui į ekraną.

```
#include <iostream.h>

void main(void)
  {
    cerr << “Programos klaida”;
  }
```

Sukompiliuokite ir paleiskite šią programą. Toliau pabandykite programos išvedimą nusiųsti į failą, panaudojant išvesties peradresavimo operatorių:

```
C:\ CERR >FILENAME.EXT <ENTER>
```

Operacinė sistema neleis išvedimo peradresuoti ir pranešimas pasirodys tik ekrane.

Išvedimo formato valdymas

Kad teisingai išvedant išdėstyti skaičius, galima įterpti tarpo (space) simbolius. Bet tai galima padaryti ir su modifikatoriumi setw (pločio nustatymas). Jo pagalba nurodomas minimalus simbolių

kiekis, kurį skaičius užims. Programa SETW.CPP panaudoja setw pločio parinkimui. Tam papildomai reikia prijungti failą iomanip.h:

```
#include <iostream.h>
#include <iomanip.h>

void main(void)
  {
    cout << "Mano skaičius lygus" << setw(3) << 1001 << end1;
    cout << "Mano skaičius lygus" << setw(4) << 1001 << end1;
    cout << "Mano skaičius lygus" << setw(5) << 1001 << end1;
   cout << "Mano skaičius lygus" << setw(6) << 1001 << end1;
  }
```

Kai sukompiliuosite ir paleisite šitą programą, ekrane pamatysite:

```
C:\ SETW <ENTER>
Mano skaičius lygus 1001
Mano skaičius lygus 1001
Mano skaičius lygus  1001
Mano skaičius lygus   1001
```

Su setw nurodomas minimalus pozicijų kiekis, kurį skaičius užims. Prieš tai esančioje programoje modifikatorius setw(3) nurodė tris simbolius. Tačiau skaičiui 1001 reikia daugiau simbolių, todėl cout panaudojo realiai reikalingą simbolių kiekį, kuris duotu atveju yra keturi. Verta paminėti, kad naudojant setw pločio parinkimui, nurodytas plotis tinka tik vieno skaičiaus išvedimui. Jei jums reikia nurodyti keleto skaičių plotį, jūs turite panaudoti setw keletą kartų.

Ką jums būtina žinoti?

Įsitikinkite, kad įsisavinote sekančius dalykus:

1) išėjimo srautas cout leidžia išvesti simbolius ir skaičius.
2) Naudodama specialius simbolius išėjimo srauto viduje, programa gali nurodyti naują eilutę, tabuliaciją ir kitas specialias galimybes.
3) Kursoriaus perkėlimui į naujos eilutės pradžią programos gali sukurti naują eilutę, panaudodamos simbolį \n arba modifikatorių endl.
4) Modifikatoriai dec, oct ir hex leidžia programoms išvesti reikšmes dešimtainiame, aštuoniatainiame ir šešioliktainiame pavidaluose.
5) Naudodamos išėjimo srautą cerr, programos gali įrašyti pranešimus į standartinį operacinės sistemos klaidų įrenginį (displėjaus ekraną).
6) Modifikatoriaus setw pagalba galima valdyti skaičių išvedimo plotį.

3 pamoka

Kintamieji

Programose dažnai reikia saugoti informaciją. Tai atliekama kompiuterio atmintyje kintamuosius talpinant ląstelėse. Šios pamokos pabaigoje įsisavinsite sekančius pagrindinius teiginius:

1. Jūs turite apibrėžti kintamuosius, kuriuos naudosite programoje, kompiliatoriui pranešdami jų vardus ir vaizdavimo formas (tipus).
2. Tam, kad kintamajam priskirti tam tikrą reikšmę, reikia naudoti C++ priskyrimo operatorių (lygybės ženklas).
3. Kintamojo reikšmės išvedimui reikia naudoti išėjimo srautą cout.
4. Aprašant kintamuosius reikia naudoti vardus, kurie išreiškia kintamųjų prasmę tam, kad programą būtų galima lengviau skaityti ir suprasti.
5. Reikia naudoti komentarus, aprašančius programos veikimą.

C++ programųose naudojami tokie kintamųjų tipai:

C++ kintamųjų tipai

Tipas	Saugoma reikšmė
Unsigned char	Reikšmių diapazonas nuo 0 iki 255.
Signed char	Reikšmių diapazonas nuo –128 iki 127.
Signed int	Reikšmių diapazonas nuo –32768 iki 32767
Unsigned int	Reikšmių diapazonas nuo 0 iki 65535
Unsigned long	Reikšmių diapazonas nuo 0 iki 4294967295
Signed long	Reikšmių diapazonas nuo -2147483648 iki 2147483647
Float	Reikšmių diapazonas nuo $-3.4*10^{-38}$ iki $3.4*10^{38}$
Double	Reikšmių diapazonas nuo $1.7*10^{-308}$ iki $1.7*10^{308}$

Toliau pateiktos rogramos fragmentas aprašo tris kintamuosius, kurių tipai int, float ir long:

```
#include <iostream.h>

void main(void)
  {
    int test_score;
    float salary;
    long distance_to_mars;
   }
```

Programa nieko nevykdo, o tik aprašo kintamuosius. Kaip matote, kiekvieno kintamojo aprašymas baigiasi kabliataškiu. Jei aprašote keletą vieno tipo kintamųjų, jų vardus galima atskirti kableliu. Operatorius - float pirmas,antras,trecias; aprašo tris kintamuosius su slankiu kableliu.

Kiekvienas sukurtas kintamasis turi turėti unikalų vardą. Kad savo programas padaryti lengviau skaitomas ir suprantamas, reikia naudoti prasmingus kintamųjų vardus. Pavyzdžiui, sekantis operatorius aprašo tris kintamuosius, kurių vardai x, y ir z:

```
int x, y, z;
```

Tarkim, kad šitie kintamieji saugo studento amžių, testo taškus ir įvertinimą, tada sekantys kintamųjų vardai bus labiau suprantami pagal savo prasmę kitiems programuotojams, skaitantiems jūsų kodą:

```
int amzius, taskai, ivertinimas;
```

Parenkant kintamųjų vardus galima naudoti raidžių ir skaičių kombinacijas, brūkšnelius (_). Pirmas kintamojo vardo simbolis turi būti raidė arba brūkšnelis. Negalima kintamojo vardo pradėti skaičiumi. Be to C++ apatinio ir viršutinio klaviatūros registrų raidės skaitomos skirtingomis.

Žodžiai, kurių negalima vartoti kintamųjų vardams

Nustatant kintamųjų vardus būtina žinoti, kad žemiau pateiktoje lentelėje išvardinti žodžiai yra C++ kalboje rezervuoti .

C++ raktiniai žodžiai

Asm	Auto	Break	Case	Catch	Char	Class	Const	Continue
Default	Delete	Do	Double	Else	Enum	Extern	Float	For
Friend	Goto	If	Inline	Int	Long	New	Operator	Private
Protected	Public	Register	Return	Short	Signed	Sizeof	Static	Struct
Switch	Template	This	Throw	Try	Typedef	Union	Unsigned	Virtual
Void	Volatile	While						

Reikšmių priskyrimas kintamiesiems

Programos vykdymo metu kintamieji saugo reikšmes. Jų priskyrimas atliekamas su C++ priskyrimo operatoriumi (lygybės ženklas). Sekantys operatoriai priskiria reikšmes keliems skirtingiems kintamiesiems. Atkreipkite dėmesį į po kiekvieno operatoriaus sekantį kabliataškį:

```
age = 32;
salary = 25000.75;
distance_to_the_moon = 238857;
```

Pastaba: kintamiesiems priskiriamos reikšmės negali turėti kablelių (pvz. 25,000.75 ir 238,857). Jei jūs padėsite kablelį, C++ kompiliatorius išves pranešimą apie sintaksės klaidas.
Sekančios programos fragmentas iš pradžių aprašo kintamuosius, o po to priskiria jiems reikšmes.

```
#include <iostream.h>

void main(void)
  {
     int age;
     float salary:
     long distance_to_the_moon;

     age=32;
     salary= 25000.75;
     distance_to_the_moon=238857;
   }
```

Aprašant kintamąjį dažnai patogu jam iš kart suteikti pradinę reikšmę.

```
int age = 32
float salary = 25000.75;
long distance_to_the_moon = 23885
```

Po priskyrimo programa gali naudoti tą reikšmę kreipdamasi į kintamąjį. Programa SHOWVARS.CPP priskiria reikšmę trims kintamiesiems, o paskui išveda kiekvieno iš jų reikšmę panaudodama cout:

```
#include <iostream.h>
void main(void)
  {
     int age  = 32;
     float salary = 25000.75;
     long distance_to_the_moon = 238857;

     cout << "Tarnautojui " << age << " metai " << end1;
     cout << "Tarnautojo uždarbis yra $" << salary << end1;
     cout << "Nuo žemės iki mėnulio yra" << distance_to_the_moon
          << " mylios" << end1;
  }
```

Pastaba: paskutinis operatorius cout netelpa vienoje eilutėje. Tokiu atveju programa paprasčiausiai perkelia žodžius į sekančią eilutę. Jūs galite atlikti tokį perkėlimą, kadangi C++ operatoriaus pabaigai parodyti naudoja kabliataškį. Jei jums būtina perkelti eilutę, pasistenkite nenutraukti simbolių eilutės (tai tekstas tarp dvigubų kabučių), nukeltai daliai panaudokite papildomą << ženklą, kaip parodyta aukščiau.
Kai sukompiliuosite ir paleisite programą, ekrane turite pamatyti:

```
C:\> SHOWVARS <ENTER>

Tarnautojui 32 metai
```

Tarnautojo uždarbis yra $25000.75
Nuo žemės iki mėnulio yra 238857 mylios

Kintamojo reikšmių diapazono viršijimas

Nuo kintamojo tipo priklauso reikšmių diapazonas, kurį jis gali saugoti. Pavyzdžiui, *signed int* tipo kintamasis gali saugoti reikšmes diapazone nuo -32768 iki 32767. Jei jūs priskiriate kintamajam reikšmę, kuri neįeina į ši diapazoną, atsiranda perpildymo klaida. Pavyzdžiui programa OVERFLOW.CPP iliustruoja, kaip atsiranda klaida dėl kintamojo reikšmių diapazono viršijimo. Kaip matote, programa priskiria reikšmes, kurios nebetelpa į kintamojo reikšmių diapazoną:

```
#include <iostream.h>
void main(void)
 {
   int positive = 40000;
   long big_positive = 4000000000;
   char little_positive = 210;

   cout << “dabar positive sudaro “ << positive << end1;

                  Cout     << “dabar big_positive sudaro “ << big positive
                           << endl;
                  cout     << “dabar little_positive sudaro “<< little_positive
                           << endl;
         }
```

Kai sukompiliuosite ir paleisite šią programą , ekrane atsiras sekantis užrašas:

```
Dabar positive sudaro –25536
Dabar big_posityve sudaro –294967296
Dabar little_posityve sudaro T
```

Kaip matome , programa int , long ir char tipo kintamiesiems priskiria reikšmes, kurios randasi ne kiekvieno tipo saugojimo reikšmių diapazone. Atsiranda perpildymo klaida. Dirbant su kintamaisiais būtina atsiminti reikšmių diapazonus, kuriuos gali saugoti kiekvienas kintamasis. Perpildymo klaidos yra sunkiai pastebimos, todėl jas yra sunku surasti ir ištaisyti. Atkreipkite dėmesį į little_posityve reikšmę , kurią programa išved į ekraną. Kadangi šis kintamasis yra char tipo , tai išėjimo srautas cout stengiasi išvesti jo reikšmę simboliniame pavidale (210).

Tikslumas

Reikia žinoti, kad kompiuteris negali užtikrinti neriboto tikslumo. Pvz., dirbant su slankaus kablelio skaičiais kompiuteris ne visada gali priimti skaičių su reikiamu tikslumu.Todėl galimos apvalinimo klaidos, kurias nelengva aptikti.
Programa PRECISE.CPP priskiria float ir double tipų kintamiesiems reikšmes, šiek tiek mažesnes už 0,5. Deja rezultatas bus 0,5.

```
#include <iostream.h>

void main(void)
         {
                  float f_not_half = 0.49999990;
                  double d_not_half = 0.49999990;
                  cout     << “tipo float reikšmė 0,49999990 lygi “
                           << f_not_half << endl;
```

```
        cout    << "tipo double reikšmė 0,49999990 lygi"
                << d_not_half << endl;
}
```

Kai sukompiliuosite ir paleisite šią programą, jūsų ekrane atsiras sekantis užrašas:

```
Tipo float reikšmė 0,49999990 lygi 0.5
Tipo double reikšmė 0,49999990 lygi 0.5
```

Kaip matote , reikšmės , priskirtos kintamiesiems , ir reikšmės , kurias kintamieji realiai turi , nėra tos pačios. Tokios apvalinimo klaidos įvyksta todėl, kad kompiuteris operuoja su fiksuotu skilčių skaičiumi vaizduojamais skaičiais.

Komentarų panaudojimas

Daugybė programoje esančių operatorių gali ją padaryti sunkiai skaitomą. Kad ji taptų aiškesnė reikia:

- Naudoti prasmingus kintamųjų vardus;
- Naudoti tam tikrą teksto formavimo techniką;
- Naudoti tuščias eilutes tam , kad būtų atskirti nepriklausomi operatoriai;
- naudoti komentarus , aiškinančius programos darbą.

Komentarai C++ kalboje yra formuojami su simboliais (//) pradžioje

```
//čia komentarai, arba
 /* tai komentaras */
```

Kai C++ kompiliatorius suranda //, jis ignoruoja visą po to einantį eilutės tekstą. Bet kuriuo atveju, kiekvienos programos pradžioje stenkitės parašyti komentarus kurie nurodytų , kas parašė programą, kada ir ką ji veikia:

```
// programa : BUDGET.CPP
// programavo : Jonas Jonaitis
// sukūrimo data : 1-10-96
//
// tikslas: Skaitmeninio voltmetro valdymo programa
```

Ką būtina prisiminti?

- tam , kad programoje galėtumėte naudotis kintamaisiais reikia jiems pirmiausia priskirti tipą ir vardą;
- kintamųjų vardai turi būti unikalūs ir suprantami pagal prasmę;
- kintamųjų vardai negali prasidėti skaičiumi;
- C++ kalboje apatinio ir viršutinio klavitūros registrų raidės yra laikomos skirtingomis.
- kintamojo tipas nusako reikšmę, kurią jis gali turėti. Kintamųjų tipai yra char, int, float , long ir double;
- komentarai palengvina programos skaitymą ir aiškina jos darbą. Komentarai C++ kalboje formuojami su // (slash) ir /* */.

4 Pamoka

Pagrindinės matematinės operacijos

Lentelėje pateikiamos pagrindines C++ kalboje naudojamos operacijas.

Veiksmas	Paskirtis	Pavyzdys
+	Sudėtis	Total = cost + tax
-	Atimtis	Change = payment – total;
*	Daugyba	Tax = cost * tax_rate;
/	Dalyba	Average = total / count;

Programa SHOWMATH.CPP naudoja “cout” tam , kad išvestų keletą paprastų aritmetinių operacijų rezultatų :

```
#include <iostream.h>

void main(void)
        {
                cout << “5+7 =” << 5+7 << endl;
                cout << “12-7 = “ << 12-7 << endl;
                cout << “ 1.2345 * 2 = “ << 1.2345 * 2 << endl;
                cout << “ 15/3 = “ << 15/3 << endl;
        }
```

Atkreipkite dėmesį , kad kiekvienas išsireiškimas pirmiausiai atsiranda kabutėse. Tai leidžia išvesti simbolius (pav., 5+7 =) į ekraną. Po to programa išveda rezultatą ir naujos eilutės simbolius. Kai sukompiliuosite ir paleisite programą , jūsų ekrane pasirodys toks tekstas:

```
C:\> showmath <enter>
5 + 7 = 12
12 – 7 = 5
1.2345 * 2 = 2.469
15 / 3 = 5
```

Duotu atveju programa vykdo aritmetines operacijas, naudodama tik pastovias reikšmes. Programa MATHVARS.CPP vykdo aritmetines operacijas, naudodama kintamuosius:

```
#include <iostream.h>
void main(void)
        {
                float coast = 15.50;            //pirkinio kaina
                float sales_tax                 //pardavimo mokestis 6%
                float amount_paid               //pirkėjo pinigai
                float tax, cieling, total;       //pardavimo mokestis
                                                //grąža pirkėjui ir bendra sąskaita

                tax = coast * sales_tax;
                total = cost + tax;
                cieling = amount_paid – total;

                cout << “pirkinio kaina: $ “ << cost <<
                        “\ t mokestis: $” << tax << “\t bendra sąskaita: $” << total << end1;
                cout << “grąža pirkėjui: $” << cieling << endl;
        }
```

Duotu atveju programa naudoja tik kintamuosius su slankiu kableliu. Kai sukompiliuosite ir paleisite programą , jūsų ekrane pasirodys toks tekstas:

```
C:\> MATHVARS <enter>
Pirkinio kaina: $15.5 Mokestis: $0.93 Bendra sąskaita: $16.43
Grąža pirkėjui: $3.57
```

Dažnai tenka naudotis kintamojo reikšmės inkrementavimu (1 pridėjimu).

```
Count = count + 1;
```

Duotoju atveju programa pirmiausiai paima count reikšmę , o paskiau prideda vienetą. Rezultatas patalpinamas į count. Programa INTCOUNT.CPP panaudoja proskyrimo operatorių tam, kad padidintų kintamojo count reikšmę vienetu (priskiriant kintamajam reikšmę 1001):

```
#include <iostream.h>

void main (void)
        {
                int count = 1000;
                cout << "Padinė count reikšmė lygi" << count << endl;
                count =  count + 1;
                count << "Gutinė count reikšmė lygi" << count <<endl;
        }
```

Kai sukompiliuosite ir paleisite programą , jūsų ekrane pasirodys toks tekstas:

```
C:/> INTCOUNT <enter>
Pradinė count reikšmė lygi 1000
Galutinė count reikšmė lygi 1001
```

C++ kalboje yra didinimo vienetu komanda - ++.

```
Count = count + 1;                    count ++;
```

Programa INC_OP.CPP padidina count reikšmę vienetu:

```
#include <iostream.h>

void main(void)
        {
                int count = 1000;

                count << "pirminė count reikšmė lygi " << count << endl;
                count++;
                count << "galutin4 count reikšmė lygi "<< count << endl;
        }
```

Ši programa dirba taip pat , kaip ir INCOUNT.CPP, kuri panaudojo priskyrimo operatorių tam, kad padidintų kintamojo reikšmę. Kai C++ sutinka didinimo komandą, tai pirmiausiai yra pasiimama kintamojo reikšmė, prideda prie jos vienetą, o paskiau rezultatą įrašo atgal į kintamojo vietą.

Programos gali patalpinti didinimo operatorių ++ iki arba po kintamojo:

```
++variable;                    variable++;
```

Kai operatorius ++ randasi prieš kintamąjį , tai jis yra vadinamas priešdidinimo operatoriumi. Antras operatorius ++ randasi po kintamojo ir yra vadinamas podidinimo operatoriumi. C++ kalba juos supranta skirtingai. Užrašas

```
Current_count = count++;
```

reiškia tokią įvykių seką:
```
Current_count = count;
count = count +1;
```

O užrašas
```
Current_count = ++count;
```

reiškia tokią įvykių seką:
```
count = count +1;
Current_count = count;
```

Programa PRE_POST.CPP vaizduoja prieš- ir po- didinimo operacijų naudojimą:

```
#include <iostream.h>

void main(void)
        {
                int small_count = 0;
                int big_count = 1000;

                cout << "small_count lygus" << small_count << endl;
                cout << "small_count++ padaro "<< small_count++ << endl;
                cout << "galutinė small_count reikšmė lygi"
                        << small_count << endl;
                cout << "big_count lygu "<< big_count << endl;
                cout << "++big_count padaro " << ++big_count << endl;
                cout << "galutinė  big_count reikšmė lygi "<< big_count
                     << endl;
        }
```

Kai j sukompiliuosite ir paleisite šią programą , jūsų ekrane atsiras sekantis užrašas:

```
C:\>  PRE_POST <enter>
Small_count lygus 0
Small_count++ padaro 0
Galutinė small_count reikšmė lygi 1
Big_count lygi 1000
++big_count padaro 1001
galutinė big_count reikšmė lygi 1001
```

Operatorius -- reiškia mažinimą vienetu. Kaip ir padidinimo operacijose C++ kalba turi prieš- ir po-mažinimą. Programa DECCOUNT.CPP vaizduoja mažinimo operatoriaus panaudojimą C++ kalboje.

```
#include <iostream.h>

void main(void)
        {
                int small_count = 0;
                int big_count = 1000;

                cout << "small_count lygus" << small_count << endl;
                cout << " small_count – padaro " << small_count-- << endl;
                cout << "galutinė small_count reikšmė lygi "
                      << " small_count << endl;

                cout << " big_count lygus"<< big_count << endl;
                cout << " --big_count padaro" << --big_count << endl;
```

```
                    << big_count << endl;
        }
```

Kai sukompiliuosite ir paleisite šią programą ekrane pasirodys sekantis užrašas:

```
C:\> DECCOUNT <enter>
Small_count lygus 0
Small_count – padaro 0
Galutinė small_count reikšmė lygi –1
Big_count lygus 1000
--big_count padaro 999
galutinė big_count reikšmė lygi 999
```

Kaip matote prieš- ir po- mažinimo operatoriai C++ kalboje dirba analogiškai kaip ir didinimo operacijose , tik skirtumas yra tas , kad čia kintamojo reikšmė mažinama 1.

Kiti C++ operatoriai

C++ programose galima panaudoti ir lentelėje pateiktas operacijas:

C++ programose sutinkamos operacijos

Operacija	Funkcija
%	Sveikų skaičių liekana po dalybos
~	Bitų inversija
&	Pobitinis loginis IR
|	Pobitinis loginis ARBA
^	Pobitinė loginė suma moduliu 2
<<	Postūmis į kairę. Perstumia bitus į kairę per nurodytą skilčių skaičių
>>	Postūmis į dešinę. Perstumia bitus į dešinę per nurodytą skilčių skaičių

Operacijų vykdymo eilė

Aprašant aritmetinius veiksmus C++ programose būtina žinoti, kad jos C++ kalboje vykdomos tam tikra tvarka. Pavzdžiui sumavimas vykdomas po sandaugos:

```
result = 5+ 2 * 3;
```

Priklausomai nuo veiksmų eiliškumo rezultatas gali būti:

```
result = 5 + 2 * 3;          result = 5 + 2 * 3;
       = 7 * 3;                     = 5 + 6;
       = 21;                        = 11;
```

Kad išvengti painiavos, C++ kalboje kiekvienai operacijai suteikiamas prioritetas, nustatantis veiksmų eilę. Kadangi C++ vykdo aritmetines operacijas tam tikra tvarka, tai ir jūsų programos atitinkama tvarka vykdys veiksmus. Lentelėje pateikta C++ operacijų vykdymo tvarka. Lentelės pradžioje esančios operacijos turi aukštesnį prioritetą. Operacijos esančios toje pačioje lentelės dalyje turi vienodą prioritetą. Daugelio operacijų, pateiktų lentelėje jūs dar nežinote, bet galite į tai nekreipti jokio dėmesio. Baigdami studijuoti šią knygą, jas visas žinosite ir galėsite naudoti programuojant.

C++ operacijų vykdymo prioritetai

Operacija	Apibūdinimas	Pavyzdys
::	Kintamojo apibrėžimo sritis	class_name::class_member_name::variable_name
::	Globalaus kintamojo apibrėžimas	::variable_name
.	Elemento išrinkimas	object.member_name
->	Elemento išrinkimas	pointer-member_name
[]	Indeksacija	pointer[element]

()	Funkcijos kvietimas	expression(parameters)
()	Reiškinio rezultatas	type(parameters)
Sizeof	Objekto dydis	sizeof expression
Sizeof	Tipo dydis	sizeof(type)
++	Inkrementavimas po	variable++
++	Inkrementavimas prieš	++variable
--	Dekrementavimas po	variable--
--	Dekrementavimas prieš	--variable
&	Objekto adresas	&variable
*	Pervardinimas	*pointer
New	Sukūrimas (išdėstymas)	new type
Delete	Panaikinimas (ištrynimas)	delete pointer
delete[]	Masyvo panaikinimas	delete pointer
~	Papildymas	~expression
!	Inversija	!expression
+	Pridėti	+1
-	Atimti	-1
.*	Elemento išrinkimas	object.*pointer
->	Elemento išrinkimas	object->*pointer
*	Daugyba	expression*expression
/	Dalyba	expression/expression
%	Liekana po dalybos	expression%expression
+	Sudėtis	expression+expression
-	Atimtis	expression-expression

Veiksmų vykdymo eilė

C++ priskiria operacijoms skirtingus prioritetus, nustatančius veiksmų eilę. Tačiau kartais reikia kitokios veiksmų atlikimo eilės, negu nustato C++. Pavyzdžiui reikia atlikti sudėtį prieš daugybą:

```
cost = price_a + price_b * 1.06;
```

Šiuo atveju C++ pirmiausiai atliks daugybą (*price_b * 1.06*), o po to pridės reikšmę *price_a*. Veiksmų atlikimo tvarką jūs galite nustatyti, naudodami skliaustus. C++ visada pirmiausiai vykdo skliaustuose esančius veiksmus. Pavyzdžiui:

```
result = (2 + 3) * (3 + 4);
```

C++ atliks veiksmus tokia tvarka:

```
result = (2 + 3) * (3 + 4);
       = (5) * (3 + 4);
       = 5 * (7);
       = 5 * 7;
       = 35;
```

Taip apskliausdami reiškinius, jūs galite valdyti C++ aritmetinių veiksmų vykdymo eilę.

Klaidos aritmetinėse operacijose

Priskiriant kintamajam reikšmę, išeinančią iš jo tipo reikšmių diapazono ribų, atsiranda perpildymo klaida. Atliekant aritmetinius veiksmus, būtina prisiminti apie perpildymo klaidos galimybę. Pavyzdžiui, sekanti programa MATHOVER.CPP daugina 200 ir 300 bei rezultatą priskiria signed *int* tipo kintamajam. Tačiau kadangi daugybos rezultatas (60000) viršija didžiausią *int* tipo kintamojo reikšmę (32767), atsiranda klaida:

```
#include <iostream.h>
void main (void)
        {
        signed int result;
        result = 200 * 300;
        cout << "200 * 300 = "<< result <<end1;
        }
```

Sukompiliavus ir paleidus programą, ekrane jūs pamatysite:

```
C:\> MATHOVER <ENTER>
200 * 300 = -5536
```

Ką būtina žinoti?

Įsitikinkite, kad išmokote:

- ✓ C++ naudoja operatorius +, - ,* , / sudėčiai, atimčiai, daugybai ir dalybai.
- ✓ C++ galima naudoti pre- ir post- didinimo ir mažinimo vienetu operacijas.
- ✓ Pre- operacijos nurodo, kad reikia pirmiausiai padidinti (arba sumažinti) reikšmę, o po to ją priskirti kintamajam.
- ✓ Post- operacijos nurodo, kad reikia pirmiausiai reikšmę priskirti kintamajam, o po to ją padidinti (arba sumažinti).
- ✓ Kiekvienai operacijai suteikiamas prioritetas tam, kad užtikrinti teisingą veiksmų atlikimo tvarką,
- ✓ Norėdami pakeisti veiksmų atlikimo tvarką, naudokite skliaustus. C++ pirmiausiai vykdo veiksmus, esančius skliaustuose.

5 pamoka

Įvestis iš klaviatūros

Šioje pamokoje išmoksite naudoti įvedimo srautą *cin* informacijos įvedimui iš klaviatūros. Naudodami *cin*, jūs nurodote vieną ar kelis kintamuosius, kuriems *cin* priskirs įvestas reikšmes. Pamokos pabaigoje jūs įsisavinsite:

- priskyrę kintamajam reikšmę (įvestą iš klaviatūros) su *cin*, , jūs ją galite naudoti taip pat, kaip ir priskyrę su operatoriumi =.
- Naudodami *cin* įvedimui, stenkitės išvengti perpildymo klaidų ir klaidų, atsirandančių vartotojui įvedus klaidingo tipo reikšmę.

Naudojant *cin*, bus naudojamas ištraukimo operatorius (>>).

Pirmoji pažintis su cin

Įėjimo srautas *cin* aprašo įvedimą iš klaviatūros. Kada programoje naudojamas *cin*, jūs turite nurodyti kintamąjį, kuriam *cin* priskirs įvestus iš klaviatūros duomenis. Programa FIRSTCIN.CPP naudoja *cin* klaviatūra surinkto skaičiaus nuskaitymui. Programa priskiria įvestą skaičių kintamajam *number*, o po to išveda į ekraną jo reikšmę, naudodama *cout*:

```
#include <iostream.h>

void main(void)
{
int number; // skaičius, kuris bus nuskaitomas iš klaviatūros
```

```
cout << “Įveskite skaičių ir nuspauskite ENTER:”;
cin >> number;
cout << “Jūs įvedėte skaičių” << number << end1;
}
```

Kai sukompiliuosite ir paleisite šią programą, ekrane pasirodys pranešimas prašantis įvesti skaičių. Jį įvedus ir nuspaudus ENTER, programa priskirs įvestą reikšmę kintamajam *number*. Po to naudodamas *cout* ir programa išves įvesto skaičiaus reikšmę. Kita programa TWONBRS.CPP prašo dviejų skaičių. Jie priskiriami kintamiesiems *first* ir *second.* Paskui, naudodama *cout,* programa juos išves į ekraną:

```
#include <iostream.h>
void main(void)
{
int first, second; // skaičiai, įvedami iš klaviatūros

cout << ”įveskite du skaičius ir nuspauskite ENTER:”;
cin >> first >> second;
cout << “buvo įvesti skaičiai” <<
        first << “ir” << second <<end1;
}
```

Atkreipkite dėmesį į dviejų operatorių panaudojimą:

```
cin >> first >> second;
```

Šiuo atveju *cin* priskirs pirmą įvestą reikšmę kintamajam *first*, antrąją - *second*. Jeigu programai reikalinga trečioji reikšmė, jūs galite naudoti trečią operatorių:

```
cin >> first >> second >> third;
```

Jeigu naudojate *cin* skaičių įvedimui iš klaviatūros, skaičiaus pradžiai ir galui nustatyti *cin* naudoja pirmą tuščią simbolį (tarpą, tabuliaciją, markerio grąžinimą į eilutės pradžią). Paeksperimentuokite su programa TWONBRS.CPP, atskirdami skaičius tuščiais simboliais.

Įvedimas iš klaviatūros su cin

Nuskaitymui iš klaviatūros galima naudoti *cin*. Tam būtina nurodyti kintamuosius, kuriems priskiriami įvedami duomenys.

```
cin >> some_variable;
```

cin paima duomenis iš įvedamo srauto ir priskiria juos kintamiesiems.

Perpildymo klaidos

Jei jūsų programos vykdo įvedimą su *cin*, reikia apsisaugoti nuo klaidų, atsirandančių, vartotojui įvedus neteisingą skaičių. Pavyzdžiui, paleiskite programą FIRSTCIN.CPP ir įveskite skaičių 1000000. Šiuo atveju programa negalės išvesti į ekraną skaičiaus 1000000, nes jis per didelis *int* tipo kintamajam.

Jeigu jūs atidžiai išnagrinėjote programą FIRSTCIN.CPP, tai atkreipėte dėmesį į tai, kad *cin* priskiria įvestą skaičių int tipo kintamajam. Kaip sužinojote 4 – oje pamokoje, *int* tipo kintamieji gali būti intervale [-32768;+32767]. Kadangi *int* tipo kintamasis negali saugoti reikšmės 1000000, atsiras

klaida. Pabandykite paleisti programą daugiau kartų, įvesdami teigiamus ir neigiamus skaičius, ir atkreipkite dėmesį į klaidas, atsirandančias, įvedant skaičius, nepriklausančius intervalui, kuriame apibrėžti *int* tipo kintamieji.

Tipų nesutapimas

Kaip jau buvo minėta, FIRSTCIN.CPP programa parašyta taip, kad vartotojas turi įvesti skaičių diapazone nuo –32768 iki 32767. Jeigu jis įveda raidę ar kokį kitą simbolį – mes turėsime tipų neatitikimo klaidą. Kitaip tariant, programa "laukė" tipo *int* reikšmės, o vartotojas įvedė *char* tipo reikšmę. Paleidus programą ir įvedus raides ABC, bus klaida.

Tą patį atliekant su programa TWONBRS, taip pat turėsime klaidas. Vėliau išmoksite organizuoti įvedimą taip, kad klaidos tikimybė būtų minimali. Kol kas tik žinokite, kad klaida yra galima.

Simbolių skaitymas

Abi anksčiau nagrinėtos programos naudojo *cin* sveikų skaičių priskyrimui *int* tipo kintamiesiems. Sekanti programa CIN_CHAR.CPP naudoja *cin* simbolių įvedimui iš klaviatūros. Kaip matote programa simbolius priskiria *char* tipo kintamiesiems:

```
#include <iostream.h>

void main(void)
{
char letter;

cout << "įveskite bet kokį simbolį ir nuspauskite ENTER:";
cin >> letter;
cout << "buvo įvestas simbolis" << letter << end1;
}
```

Sukompiliuokite ir paeksperimentuokite su programa, įvesdami daugiau nei vieną simbolį. Stebėkite programos reakciją į tai. Jūs pastebėsite, kad programa dirba tik su vienu simboliu.

Žodžių skaitymas iš klaviatūros

Antroje knygos dalyje jūs išmoksite kintamajam priskirti žodžius ar net teksto eilutę. Taip pat sužinosite, kaip naudoti *cin* žodžių ir eilučių skaitymui. O dabar galite parašyti programą, nuskaitančią iš klaviatūros *float* ir *long* tipo reikšmes. Pavyzdžiui, sekanti programa CIN_LONG.CPP naudoja *cin long* tipo reikšmei nuskaityti:

```
#include <iostream.h>
void main(void)
{
long value;
cout << "įveskite didelį skaičių ir nuspauskite ENTER:"
cin >> value;
cout << "buvo įvestas sakičius" << value << end1;
}
```

Paeksperimentuokite su programa, įvesdami labai didelius, neigiamus ir teigiamus skaičius.

Ką jūs turite žinoti?

Šioje pamokoje įvedimui iš klaviatūros išmokote naudoti *cin*. Kaip jau žinote, jeigu naudojate *cin*, jūs turite nurodyti kintamuosius, kuriems priskiriami įvedami duomenys.

- ✓ C++ suteikia galimybę organizuoti įvedimą iš klaviatūros, naudojant *cin*.
- ✓ Jei jūs įvedimui naudojate *cin*, turite nurodyti kintamuosius, kuriems bus priskiriamos įvedamos reikšmės.
- ✓ Įvedamos reikšmės priskyrimui, reikia naudoti ištraukimo operatorių >>.
- ✓ Naudojant *cin* kelių reikšmių įvedimui, jos atskiriamos tuščiais simboliais.
- ✓ Jei vartotojas įveda neteisingus duomenis gali kilti perpildymo ir tipų neatitikimo klaidos.

6 pamoka

Programa yra komandų seka. Visos anksčiau parašytos C++ programos vykdo komandas nuosekliai. Bet gali tekti vykdyti atskirą komandų grupę, esant patenkintai tam tikrai sąlygai, ir priešingai - kai sąlyga nepatenkinta, reikia vykdyti kitą komandų grupę. Kitaip tariant reikia, kad programa darytų sprendimus ir atitinkamai į juos reaguotų. Šioje pamokoje aprašomas operatorius *if*, kuris bus naudojamas sprendžiant panašias problemas. Į pamokos pabaigoje įsisavinsite:

- C++ programos naudoja palyginimo operacijas, kad nustatytų ar du dydžiai lygūs, ar vienas didesnis, ar mažesnis už kitą;
- Sprendimo priėmimui naudojamas operatorius *if*;
- C++ operatoriai gali būti paprasti (viena operacija) arba sudėtiniai (kelios operacijos įterptos tarp figūrinių skliaustų);
- Programos naudoja operatorių *if – else* vienos operatorių grupės vykdymui, jei sąlyga patenkinta, ir kitos, jeigu sąlyga nepatenkinta;
- Kombinuojant kelis operatorius *if – else*, galima tikrinti kelias sąlygas
- Naudojant loginius C++ operatorius IR ir ARBA, galima tikrinti kelias sudėtines sąlygas.

Dviejų reikšmių palyginimas

C++ kalboje yra naudojami tokie palyginimo operatoriai:

==	abu operandai lygūs viens kitam;
!=	abu operandai nelygūs viens kitam;
>	pirmas operandas didesnis už antrą;
<	pirmas operandas mažesnis už antrą;
>=	pirmas operandas didesnis už antrą arba lygus jam;
<=	pirmas operandas mažesnis už antrą arba lygus jam.

Sulyginimo išraiškos (operandas - sulyginimo operatorius - operandas) rezultatas yra arba loginis 1-true (kai sulyginimas teisingas), arba loginis 0 false (kai sulyginimas neteisingas).

Operatorius if

If yra sąlyginis operatorius. Šio operatoriaus formatas:

```
If  (sulyginimo išraiška)  {
    paraiška 1;
    paraiška 2;
    .
    .
    .
    paraiška n;
}
```

Kai sulyginmo išraiška teisinga (loginis 1), tai paraiškų blokas, esantis iškart po operatoriaus if, yra atliekamas. Jei ši išraiška neteisinga (loginis 0), tai paraiškų blokas yra ignoruojamas.

Operatorius if-else

Junginio If-else pagalba yra aprašomi du paraiškų blokai – vienas iškart po operatoriaus if, kitas iškart po operatoriaus else. Todel, esant neteisingai sulyginimo išraiškai (loginiam 0), kompiliatorius įvertina antrąjį paraiškų bloką:

```
If   (sulyginimo išraiška)    {
      paraiška 1;
      paraiška 2;
      .
      .
      .
      paraiška n;
}
else  {
     paraiška a;
     paraiška b;
      .
      .
      .
    paraiška z;
}
```

Jeigu paraiškų bloke yra tik viena paraiška, tai ji gali būti neskliaudžiama. Jei paraiškų bloke yra daugiau nei viena paraiška, bet tas blokas nėra apskliaudžiamas, tai if ar else operatoriui bus priskirta tik pirmoji paraiška.

Persipynę if-else operatoriai

Vienas if operatorius leidžia daryti tik vieną sprendimą. Tačiau daugelyje atvejų programa turi padaryti eilę sprendimų. Tuo tiksu gali būti naudojami persipynę if-else operatoriai:

```
for ( i =-5; i<=5; i++ )
{          if( i>0 )
           {          if( i%2==0 )
                      printf(“ %d yra lyginis skaičius.\n”, i );
                      else
                      printf(“ %d yra nelyginis skaičius.\n”, i );      }
           else if ( i==0 )
           printf(“ skaičius yra nulis.\n”);
           else
           printf(“ neigiamas skaičius.\n”);
}
```

Čia yra atrenkami lyginiai, nelyginiai, lygūs nuliui ir neigiami skaičiai.

Paskyrimo ir loginiai operatoriai

Paskyrimo operatoriaus (=) formatas:

kairysis operandas = dešinysis operandas

Šios paraiškos dėka dešniojo operando reikšmė yra įrašoma į kairiajam operandui skirtą atminties vietą.

Loginiai operatoriai:

&&	loginis IR operatorius;
||	loginis ARBA operatorius;
!	loginis NE operaturius.

Loginio IR (&&) operatoriaus formatas:

išraiška1 && išraiška2

Jei abi išraiškos yra teisingos (loginiai 1-tai), tai loginis TAIP operatorius grąžina 1-tą. Bet kokiu kitu atveju yra grąžinamas 0-lis.

Pavyzdžiui, jei reikia atlikti skaičiavimus, tik tada, kai darbuotojas gauna valandinį užmokestį ir dirba daugiau nei 40 valandų per savaitę, tai šią sąlygą galima išreikšti taip:

```
if(( darbuotojo_užm==valandinis_užm ) && ( darbuotojo_užm>40 ))
išraiška;
```

Loginio ARBA (||) operatoriaus formatas:

Išraiška1 || išraiška2

Jei bent viena išraiška yra teisinga (loginis 1-tas), tai loginis ARBA operatorius grąžina 1-tą. 0-lis bus grąžinimas tik jeigu abi išraiškos yra klaidingos (loginiai 0-liai).

Operatorius ARBA yra naudojamas kai yra reikalinga tik bent viena teisinga sąlyga. Pavyzdžiui kai reikia atlikti skaičiavimus, kai transporto priemonė yra bent viena iš dviejų pateiktųjų (automobilio ir motociklo), tai šią sąlygą galima išreikšti taip:

```
if(( tr_priemonė==automob ) || ( tr_priemonė==motoc))
išraiška;
```

Loginis NE – tai neigimo operatorius. Šio operatoriaus formatas:

! išraiška

Išraiškos vertė gali būti arba logonis 1, arba loginis 0. Operatorius ! loginį 1-tą paverčia loginiu 0-liu, o loginį 0-lį loginiu 1-tu.

Neigimo operatoriaus taikymo pavyzdys:

```
If(( !jūs turite šunį)
cout<<"jūs privalote įsigyti šunį"<<endl;
```

Switch operatoriaus panaudojimas

Kai programa turi patikrinti nustatytas reikšmes, galima naudoti C++ operatorių *switch*. Jei naudojate operatorių switch, reikia nurodyti sąlygą ir po to vieną ar kelis variantus (*case*), kuriuos programa stengsis palyginti su sąlyga. Pavyzdžiui, ši programa SWITCH.CPP naudoja operatorių switch pranešimui apie žinių įvertinimą:

```
#include <iostream.h>

void main(void)
  {
        char grade = 'B';

        switch (grade)   {
            case   'A': cout  <<"Sveikiname, Jūs gavote A" << endl;
                  break;
            case 'B': cout << "Gerai , gavote B"<< endl;
                  break;
            case 'C': cout <<"Jūsų įvertinimas vos C" << endl;
                  break;
            case 'D': cout <<"Blogai, gavote D"<< endl;
                  break;
```

```
                default: cout <<"Siaubinga! Mokykitės geriau!"<< endl;
                        break;
                }
        }
```

Operatorius *switch* sudarytas iš dviejų dalių. Pirma operatoriaus *switch* dalis yra sąlyga, kuri yra rašoma po raktinio žodžio *switch.* Antra dalis yra galimi atitikimo variantai. Kai programa randa operatorių *switch* , ji pirmiausia žiūri į sąlygą, o po to stengiasi tarp visų galimų variantų rasti atitinkantį sąlygą. Jei tokia yra randama - vykdomi po jos nurodyti operatoriai. Pateiktoje programoje varianto 'B' išrinkimas atitinka sąlygą. Šiuo atveju programa išveda pranešimą, kad vartotojas gavo B. Jei nei vienas variantas netenkina sąlygos, vykdomas variantas *default.*

Ką būtina žinoti?

Iš šios pamokos sužinojote, kaip naudoti C++ operatorių *if* sąlyginiam duomenų apdorojimui, kuris leidžia programoms pačioms priimti sprendimus. Programos gali panaudoti operatorių *if* komandų rinkiniui vykdyti kai sąlyga patenkinta ir operatorių *else* kitam komandų rinkiniui vykdyti, jei sąlyga nepatenkinta.

Įsitikinkite, kad įsisavinote:

- C++ kalbos palyginimo operacijos leidžia programai patikrinti ar lygios dvi reikšmės arba ar viena reikšmė didesnė ar mažesnė už kitą;
- C++ operatorius *if* leidžia programai patikrinti sąlygą ir įvykdyti vieną ar kelis operatorius, jei sąlyga patenkinta;
- C++ operatorius *else* leidžiia įvykdyti vieną ar kelis operatorius jei sąlyga, tikrinama operatoriumi *if*, netenkinama;
- C++ priskiria reikšmę *true* jei tikrinamas dydis nelygus nuliui ir *false* jei lygus nuliui;
- Loginės operacijos **IR** (&&) ir **ARBA** (||) leidžia programoms naudoti keletą sąlygų;
- Operatorius **NE** (!)leidžia tikrinti priešingą sąlygą;
- Jeigu operatoriuje *if* arba *else* reikia atlikti keletą operatorių, tai tie operatoriai rašomi tarp figūrinių skliaustų {};
- Jeigu programoje reikia patikrinti ar sąlyga tenkinama prie kelių reikšmių, naudokite operatorių *switch*;
- Kai programa operatoriuje switch sutinka variantą (*case*), atitinkantį sąlygą, tai visi tolesni variantai priimami, kaip tenkinantys sąlygą. Naudodami operatorių *break* jūs galite nurodyti programai pertraukti operatoriaus *switch* vykdymą ir pereiti prie sekančio operatoriaus vykdymo.

7 pamoka

Operatorių kartojimas nustatytą kartų skaičių

Viena iš labiausiai naudojamų operacijų yra vieno ar kelių operatorių kartojimas. Pavyzdžiui viena programa galėtų pakartoti vieną ir tą patį operatorių tam, kad atspausdintų penkias failo kopijas, o kita galėtų pakartoti tam tikrą operatorių rinkinį 30 kartų tam, kad patikrintų pakilo ar nukrito akcijų kursas. C++ operatorius *for* leidžia pakartoti vieną ar kelis operatorius keletą kartų.

Jei programa naudoja operatorių *for* (dažnai vadinamą ciklu *for*), ji turi nurodyti kintamąjį, kuris vadinamas *valdančiuoju kintamuoju*, kuris nurodo ciklo vykdymo kartų skaičių. Pavyzdžiui šis ciklas *for* ciklo vykdymo kartų skaičių saugoja kintamąjame *count.* Šiuo atveju ciklas bus vykdomas 10 kartų.

```
for (count =1; count <=10; count++)
        operatorius;
```

Ciklas for yra sudarytas iš keturių dalių. Pirmos trys dalys valdo ciklo vykdymo kartų skaičių. Operatorius *count*=1; priskiria valdymo kintamajam pradinę reikšmę. Ciklas *for* vykdo šią komandą vieną kartą pradėjus vykdyti ciklą. Po to ciklas tikrina sąlygą *count<=10*. Jei ši sąlyga tenkinama ciklas *for* vykdo sekantį operatorių. Jei sąlyga netenkinama, ciklas baigiamas ir programa tęsia savo darbą nuo pirmo operatoriaus po ciklo. Jei sąlyga tenkinama ir ciklas vykdo savo operatorių, tai po to padidinama kintamojo *count* reikšmė vienetu, naudojant operatorių *count++*. Po to programa tikrina sąlygą *count<=10*. Jei ši sąlyga vis dar tenkinama, tai vėl vykdomas vidinis ciklo operatorius, didinamas ir tikrinamas kintamasis *count*.

```
For (count = 1;  count <=10;   count++)
     ──────┼──────  ──────┼──────  ──┼──────
   priskyrimas    tikrinimas   didinimas
```

Programa FIRSTFOR.CPP naudoja ciklą *for* skaičių nuo 1 iki 100 išvedimui į displėjaus ekraną.

```
#include <iostream.h>

void main(void)
  {
        int count;
        for count =1; count <=100; count++)
                cout << count << ‘ ‘;
   }
```

Kaip matote, operatorius *for* priskiria kintamąjam count reikšmę 1. Po to ciklas tikrina ar kintamojo *count* reikšmė mažesnė arba lygi 100. Jeigu taip, ciklas vykdo atitinkamą operatorių, po to didina *count*, kartodamas tikrinimą.

Programa ASKCOUNT.CPP išveda paklausimą, prašantį įvesti skaičių, kuriuo ciklas turi baigtis. Po to programa išveda skaičius nuo vieneto iki nurodyto skaičiaus:

```
#include <iostream.h>

void main(void)
  {
        int count;
        int ending_value;
        cout <<Įveskite galutinę reikšmę ir nuspauskite Enter: “
        cin>> ending_value;
        for (count = 0; count <= ending_value; count++)
                cout<< count << ‘ ‘;
}
```

Programa ADD1_100.CPP sudeda skaičius nuo 1 iki 100. Cikle for yra panaudotas sudėtinis operatorius.

```
#include <iostream.h>

void main(void)
  {
        int count;
        int total = 0;
        for (count = 1; count <=100; count++)
          {
                cout <<”Pridedu “ <<count << “ prie “ <<total;
                total = total+ count;
                count <<” gaunasi “ << total  << endl;
           }
  }
```

Visi iki šiol nagrinėti *for* ciklai didindavo valdantyjį kintamąjį vienetu. Bet tai nėra vienintelis būdas. Programa BY_FIVES.CPP išveda kas penktą skaičių:

```
#include <iostream.h>
void main(void)
 {
        int count;
        for (count = 0; count<=100; count+=5)
          cout << count << ' ';
  }
```

Naudojant ciklą for nebūtina skaitliuką tik didinti. Programa CNT_DOWN.CPP naudoja ciklą for skaičių nuo 100 iki 1 išvedimui mažėjančia tvarka:

```
 #include <iostream.h>

void main(void)
  {
        int count;
        for count =1; count >=1; count--)
                cout << count << ' ';
  }
```

Kartais klaidų programose ciklas gali niekuomet neišpildyti baigimo sąlygos ir suksis amžinai (kol nepertrauksite programos). Tokie nesibaigiantys ciklai yra vadinami begaliniais. Pavyzdžiui , šis operatorius *for* sukuria begalinį ciklą:

```
for (count = 0; count <100; wrong_variable++)
 // operatoriai
```

Kaip matote, ciklo valdantysis kintamasis yra *count*. Tačiau ciklo didinimo dalyje programa didina ne tą kintamąjį. Dėl to programa niekada nedidina kintamojo *count*, ir jis niekuomet nebus didesnis ar lygus 100. Dėl to šis ciklas virsta nesibaigiančiu.

Operatorius while

Ciklas *for* leidžia kartoti vieną ar kelis operatorius nustatytą skaičių kartų. Tačiau dažnai reikia kartoti vieną ar kelis operatorius, kol išpildyta tam tikra sąlyga. Tokiais atvejais galima naudoti ciklą *while*. Bendras operatoriaus *while* formatas atrodo taip:

```
while (sąlyga_išpildyta)
        operatorius;
```

Jei programa sutinka operatorių *while*, ji tikrina nustatytą sąlygą. Jei sąlyga patenkinta, programa vykdo ciklo *while* operatorius. Po paskutinio operatoriaus įvykdymo, ciklas while vėl tikrina sąlygą. Jeigu sąlyga vis dar tenkinama, kartojami ciklo operatoriai ir procesas kartojasi. Kai sąlyga galų gale tampa netenkinama, ciklas baigiamas ir toliau vykdomas pirmas operatorius po ciklo. Sekanti programa prašo įvesti T jei taip ir N jei ne. Po to programa naudoja ciklą *while* ir nuskaito simbolius iš klaviatūros, kol vartotojas neįves T arba N, jei simbolis nėra nei T nei N, programa formuoja garsinį signalą irašydama garso simbolį '\a' į išvedimo srautą *cout:*

```
#include <iostream.h>
void main (void)
 {
        int done = 0; // nustatoma reikšmė true, jei ivesta T arba N
        char letter;
        while (!done)
          {
```

```
cout << "\nĮveskite T arba N "
     <<" ir paspauskite Enter noredami tęsti: ";
cin >> letter;
if ((letter == 'T') || (letter=='t'))
  done =1;
else if ((letter=='N') || (letter=='n')
  done = 1;
else cout << '\a';  // išduoti garsinį signalą, dėl neteisingo
                    // simbolio
                          }
                     cout <<" Jūs įvedėte raidę " << letter << endl;
                    }
```

Kaip matote, ciklas *while* taip pat palaiko keletą operatorių sugrupuotų tarp figūrinių skliaustų. Šiuo atveju programa ciklo valdymui naudoja kintamąjį *done*. Kol programa nesibaigs (kol vartotojas nepaspaus T arba N) ciklas bus vykdomas. Kai tik įvedamas simbolis T arba N kintamąjam *done* priskiriamas vienetas ir ciklas sutabdomas.

Jūs jau žinote, kad ciklas *while* leidžia programai pakartoti rinkinį operatorių tiek kartų, kol yra patenkinama užduota sąlyga. Kai programa sutinka operatorių *while*, ji iš pradžių įvertina užduotą sąlygą. Jeigu sąlyga yra teisinga, programa įeina į ciklą. Jeigu sąlyga klaidinga, operatoriai ciklo *while* viduje yra nevykdomi. Programose galima situacija kada operatorių rinkinys turi būti vykdomas nors vieną kartą. Tokiu atveju galima naudoti ciklą *do while*:

```
do{
    operatoriai;
} while (sąlyga_teisinga);
```

Kai programa sutinka ciklą *do while,* ji įeina į jį. Po to programa įvertina nustatytą sąlygą. Jeigu sąlyga teisinga, programa grįžta į ciklo pradžią.

```
do{ <----------------------------
    operatoriai;                 |
} while (sąlyga_teisinga); ------|
```

Jeigu sąlyga neteisinga, programa ciklo komandų nekartos. Paprastai ciklas *do while* naudojamas meniu punktų atvaizdavimui ir pasirinkimo apdorojimui. Jums reikalinga, kad programa atvaizduotų meniu nors vieną kartą. Jeigu vartotojas renkasi kokį nors meniu punktą, išskyrus Quit, programa įvykdys punktą, o po to atvaizduos vėl meniu (pakartodama ciklo operatorių). Jeigu vartotojas pasirenka Quit, ciklas pasibaigia ir programa pratęs savo vykdymą nuo pirmo operatoriaus po ciklo.

Jums būtina žinoti!

Iteratyvus apdorojimas – tai programos savybė pakartoti vieną arba kelis operatorius. Šioje pamokoje buvo aprašyti iteratyvūs (arba cikliški) operatoriai C++. Kaip žinote, operatorius *for* leidžia programoms vieną ar kelis operatorius pakartoti kiek norima kartų. Naudojant operatorių *while,* programos kartoja operatorių iki tol, kol nustatyta sąlyga bus patenkinta. Pasitikslinkite ar gerai įsisavinote:

√ Operatorius C++ *for* leidžia programai pakartoti vieną ar daugiau operatorių kiek norima kartų.
√ Operatorius *for* susideda iš keturių dalių: inicializacijos, sąlygos tikrinimo, operatorių, kurie kartojasi, ir indekso didinimo.
√ Ciklas C++ *while* leidžia kartoti operatorius, kol nurodyta sąlyga teisinga.
√ Programos dažnai naudoja ciklą *while* failo nuskaitymui, kol neaptinka jo galo.
√ Operatorius C++ *do while* leidžia vykdyti vieną arba kelis operatorius nors vieną kartą .
√ Programos dažnai naudoja operatorius *do while* darbui su meniu.
√ Jeigu tikrinama sąlyga cikluose *for, while* arba *do while* tampa nepatenkinta, programa tęsia savo veiklą su sekančiu operatoriumi.

8 pamoka

Pažintis su funkcijomis

Didėjant ir sudėtingėjant programoms, iškyla poreikis skirstyti jas į dalis, vadinamas funkcijomis. Kiekviena funkcija programoje turi atlikti tam tikrą užduotį. Pavyzdžiui, jei rašote atlyginimo apmokėjimo programą, tai galite parašyti vieną funkciją, kuri skaičiuotų darbo valandas, antrą funkciją – apskaičiuojančią viršvalandinį darbo atlygį, trečią – skirtą spausdinimui ir t.t. Jeigu programai reikia atlikti tam tikrą užduotį, tai ji iškviečia tam skirtą funkciją ir perduoda į ją informaciją, kuri reikalinga apdorojimui. Šioje pamokoje jūs sužinosite, kaip sukurti ir kaip panaudoti funkcijas C++ programose, įsisavinsite tokias pagrindines koncepcijas:

- Funkcijos grupuoja susijusius operatorius užduoties įvykdymui;
- Programa iškviečia funkciją, kreipdamasi į ją vardu, už kurio yra apvalūs skliaustai, pav. *beep();*
- Po apdorojimo daugelis funkcijų grąžina tam tikro tipo reikšmę, pavyzdžiui *int* arba *float*.
- Programa perduoda parametrus (informaciją) funkcijoms, patalpindama juos į apvalius skliaustus, kurie seka po funkcijos vardo;
- C++ naudoja funkcijų prototipus grąžinamos reikšmės apibrėžimui.

Paprastai funkcija formuojama tam tikram uždaviniui spręsti. Jei ji sprendžia kelis uždavinius, tikslinga funkciją suskaidyti. Kiekviena funkcija turi turėti unikalų vardą.

Funkcija C++ pagal struktūra panaši į *main* programą. Prieš funkcijos vardą yra nurodomas jos grąžinamas kintamojo tipas. Po to seka funkcijos vardas ir skliausteliuose nurodomi perduodamų parametrų tipai.

```
Grąžinamas_tipas  Funkcijos_vardas (parametrų_sąrašas su jų tipais)
{
   kintamųjų_aprašymas;

  operatoriai;
}
```

Pavyzdžiui:

```
 void show_message(void)
{
 cout << "Labas, mokaus programuoti C++" << endl;
}
```

Žodis *void* ,apibrėžiantis funkcijos vardą, nurodo, kad funkcija nieko negrąžina. Panašiai, žodis *void* esantis skliausteliuose, nurodo, kad funkcija jokių parametrų nepriima. Programa SHOW_MSG.CPP naudoja funkciją show_message informacijos išvedimui į ekraną:

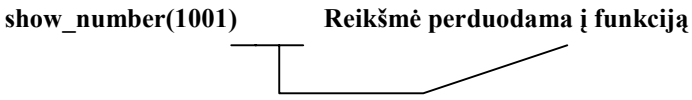
```
   #include   <iostream.h>

void show_message(void)
{
   cout   <<" Labas, mokausi  programuoti su C++" <<  endl;
}


 void main(void)
{
  cout << "Prieš funkcijos pakvietimą "  << endl;
show_message();
cout << "Grįžus iš funkcijos" << endl;
}
```

Kai programa iškviečia funkciją, ji pradeda vykdyti operatorius, esančius funkcijos viduje. Kai jos vykdymas baigiamas, grįžtama į iškviečiusiąją programą.

Funkcija gali priimti parametrus. Žemiau parodyta kaip funkcijai **show_number** perduodamas int tipo kintamasis **value**

```
    void  show_number(int value)
{
  cout << "Parametro reikšmė lygi "   << value <<endl;
}
```

Jeigu programa iškviečia funkciją s*how_number,* ji turi perduoti jai reikšmę, kaip parodyta žemiau:




```
void show_number(1001)
  {
    cout << "Parametro reikšmė lygi" << 1001 <<  end1;
   }
```

Kaip matome,kadangi C++ pakeičia parametro reikšmę, funkcija *show_number* išveda skaičių 1001, perduotą jai iš pagrindinės programos.

Programa USEPARAM.CPP naudoja funkciją *show_number* keletą kartų , kiekvieną kartą perduodama skirtingus skaičius:

```
  #include  <iostream.h>

void show_number(int value)
  {
     cout << "Parametro reikšmė lygi" << value <<  end1;
   }

void main(void)
  {
     show_number(1);
     show_number(1001);
     show_number(-532);
  }
```

Jei sukompiliuosite ir paleisite šią programą, ekrane pamatysite:

```
C:\> USEPARAM <ENTER>
 Parametro reikšmė lygi 1
 Parametro reikšmė lygi 1001
 Parametro reikšmė lygi -532
```

Kaip matome, kiekvieną kartą, kai programa iškviečia funkciją, C++ priskiria perduodamą skaičių kintamajam *value*.Paeksperimentuokite su programa, keisdami reikšmes, kurias *main* perduoda funkcijai ir atkreipkite dėmesį į rezultatus.

Kiekvienas funkcijos parametras yra tam tikro tipo.Funkcijos *show_number* atveju *value* parametras turi būti *int* tipo.Jei pabandysite perduoti funcijai kito tipo reikšmę, pavyzdžiui, su slankiu kableliu, kompiliatorius praneš apie klaidą. Galima funkcijai perduoti ir keletą reikšmių. Kiekvienam perduodamam parametrui reikia nustatyti pavadinimą ir tipą.

Programa BIGSMALL.CPP naudoja funkciją *show_big_and_little* didžiausio ir mažiausio skaičiaus, iš trijų gautų sveikųjų skaičių, išvedimui:

```
#include <iostream.h>
```

```
 void show_big_and_little ( int a, int b, int c)
    int small = a;
     int big     =a;

    if  (b > big)
       big = b;
    if (b < small)
       small = b;
    if  (c > big)
       big = c;
    if  (c < small)
      small = c;

   cout << "Didžiausia reikšmė lygi" << big << end1;
   cout << "Mažiausia reikšmė lygi" << big << end1;
}

void main(void)
{
  show_big_and_little(1, 2, 3);
  show_big_and_little(500, 0, -500);
  show_big_and_little(1001, 1001, 1001);
}
```

Kai programa iškviečia funkciją ,C++ paskiria parametrus, kaip parodyta žemiau:

```
show_big_and_little(1, 2, 3);

                        void  show_big_and_little(int a, int b, int c);
```

Jei sukompiliuosite ir paleisite šią programą, ekrane pamatysite:

```
C:\> BIGSMALL <ENTER>
Didžiausia reikšmė lygi 3
Mažiausia reikšmė lygi  1
Didžiausia reikšmė lygi 500
Mažiausia reikšmė lygi  -500
Didžiausia reikšmė lygi 1001
Mažiausia reikšmė lygi  -1001
```

Programa SHOW_EMP.CPP naudoja funkciją *show_employee*. Jai perduodamas darbuotojo amžius (tipas *int*) ir jo atlyginimas (tipas *float)*:

```
#include <iostream.h>

void show_employee(int age, float salary)
{
   cout<<"Darbuotojo amžius" << age <<"metai (metų)"
         << end1;
   cout <<"Darbuotojas gauna  $" << salary << end1;
}

void main(void)
{
   show_employee(32, 25000.00);
}
```

Kaip matome, funkcija *show_employee* nustato *int* ir *float* tipo parametrus.

Jei funkcija naudoja parametrus, ji turi nurodyti vardą ir tipą kiekvienam iš jų. Kai programa iškviečia funkciją, C++ priskiria parametrų reikšmes funkcijos parametrų vardams iš kairės į dešinę.

Daugeliu atvejų funkcijos atlieka tam tikrus skaičiavimus ir grąžina rezultatą iškvietusiai funkcijai. Kai funkcija grąžina tam tikrą reikšmę, turite C++ pranešti reikšmės tipą, pvz. - *int, float, char*. Tam prieš fukcijos pavadinimą reikia nurodyti grąžinamą tipą. Funkcija *add_values* sudeda du sveikus skaičius ir grąžina *int* tipo rezultatą:

```
  int add_values(int a, int b)
{
   int result;
   result = a + b;
   return(result);
}
```

Šiuo atveju žodis *int* , esantis prieš f-jos pavadinimą, nurodo grąžinamos reikšmės tipą. Funkcijos reikšmės grąžinimui funkcija naudoja operatorių *return.*
Kai programa sutinka operatorių *return* , ji grąžina užduotą reikšmę ir baigia funkcijos vykdymą, grąžindama valdymą iškvietusiai programai.
Programoje grąžinama reikšmė gali būti naudojama taip:
result = add_values(1, 2);

Šiuo atveju programa priskiria funkcijos grąžinamą reikšmę kintamajam *result.* Programa taip pat gali iš karto išvesti funkcijos grąžinamą reikšmę naudodama *cout* :

```
cout <<" Reikšmių suma lygi" << add_values(500,501) <<end1;
```

Ankstesnė funkcijos *add_values* realizacija naudojo tris operatorius tam , kad būtų lengviau suprasti funkcijos prasmę. Tačiau galima naudoti vienintelį operatorių *return:*

```
int  add_values(int a, int b)
  }
    return(a+b);
  }
```

Programa ADDVALUE.CPP naudoja funkciją *add_values* keleto skaičių sudėčiai:

```
#include <iostream.h>

int add_values(int a, int b)
  }
   return(a+b);
  }

void main(void)
  {
      cout <<"100+200 = "add_values(100, 200) << end1;
      cout <<"500+501 = "add_values(500, 501) << end1;
       cout <<"-1 +1      = "add_values(-1, 1) << end1;
 }
```

Paeksperimentuokite su šia programa keisdami reikšmes, kurias programa perduoda funkcijai. Pabandykite perduoti funkcijai du didelius skaičius, pvz. 20000 ir 30000.Funkcijos, grąžinančios *int* tipo reikšmę, vykdymas iššauks perpildymo klaidą.
Ne visos funkcijos grąžina *int* tipo reikšmę. Funkcija *average_value* grąžina dviejų sveikųjų skaičių vidurkį, kuris gali būti su kableliu, pvz. 3,5:

```
      float average_value(int a, int b)
      }
          return((a+b) / 2.0);
      }
```

Šiuo atveju žodis float, esantis prieš funkcijos pavadinimą, nurodo funkcijos grąžinamos reikšmės tipą.

Jei funkcija negrąžina reikšmės, būtina nustatyti *void* funkcijos tipą . Priešingu atveju reikia nustatyti funkcijos grąžinamos reikšmės tipą, pvz.*int, float, char* ir t.t. Reikšmės grąžinimui funkcija naudoja operatorių *return.* Kai programa sutinka operatorių *return* , funkcijos vykdymas baigiamas ir nurodyta reikšmė grąžinama iškvietusiai funkcijai. Galimos situacijos, kai operatorius *return* yra funkcijoje, kuri negrąžina reikšmės:

return ;

Šiuo atveju funkcijos tipas *void* ir operatorius *return* tiesiog baigia funkcijos vykdymą.

Pastaba: jei operatoriai funkcijoje bus po operatoriaus *return*, jie nebus vykdomi. Programai sutikus operatorių *return* funkcijos vykdymas baigiamas, grąžinama reišmė ir programos vykdymas tęsiamas nuo pirmo operatoriaus, sekančio po funcijos iškvietimo.

Kai funkcija grąžina reikšmę, ją iškvietusi programa gali priskirti tą reikšmę kintamajam, naudodama priskyrimo operatorių:

```
payroll_amount = payroll(employee, hours, salary) ;
```

Iškvietusi programa gali tiesiog kreiptis į funkciją. Pvz., sekantis operatorius išveda funkcijos grąžinamą reikšmę, naudodamas *cout*:

```
cout << "Darbuotojas gavo" << payroll(employee, hours, salary)
     << end1;
```

Iškvietusi funkcija taip pat gali naudoti grąžinamą reikšmę sąlygoje:

```
if (payroll(employee, hours, salary) < 500.00 )
  cout << " Šiam darbuotojui reikia padidinti ""  << end1;
```

Kaip matome, programa gali įvairiai panaudoti funkcijos grąžinamą reikšmę.

Funkcijų prototipai

Iškviesdama funkciją, C++ turi žinoti grąžinamos reikšmės tipą, o taip pat funkcijos naudojamų parametrų tipą ir kiekį. Kiekvienoje šioje pamokoje pateikiamoje programoje, funkcijos nustatymas yra prieš jos iškvietimą pagrindiniame faile.Tačiau galima funkcijas talpinti pagrindiniame faile ir po jų iškvietimo. Šiuo atveju pagrindinio failo pradžioje tenka suformuoti *funkcijų prototipus*. Bendruoju atveju funkcijos prototipas teikia informaciją apie funkcijos grąžinamos reikšmės tipą ir jos parametrus. Sekantys pavyzdžiai iliustruoja funkcijų prototipus keletui funkcijų, naudojamų šioje pamokoje:

```
void show_message(void);

void show_number(int);

void show_employee(int, float);

int add_values(int, int);

int average_value(int, int);
```

Kaip matote, funkcijos prototipas nurodo grąžinamos reikšmės tipą, o taip pat kiekvieno parametro tipą ir kiekį. Atkreipkite dėmesį į kabliataškį kiekvieno prototipo pabaigoje.

```
float average_value(int, int);
```

Šioje eilutėje float - grąžinamos reikšmės tipas, o (int, int) - parametrų tipai.

Jei programa iškviečia funkciją, kuriai C++ kompiliatorius negali surasti prototipo ar kitokio funkcijos nustatymo, pranešama apie klaidą. Programa PROTO.CPP iliustruoja funkcijos prototipo panaudojimą:

```
#include <iostream.h>

float average_value(int, int); // Funkcjos prototipas

vid main(void)
 }
     cout << "2000 ir 2 vidurkis lygus" <<
     average_value(2000, 2) << end1;
 }

float average_value(int a, int b)
 }
   return((a+b)/2.0);
 }
```

Šiuo atveju programa iškviečia funkciją *average_value* prieš funkcijos nustatymą. Jei pašalinsite funkcijos prototipą ir sukompiliusite šią programą, C++ kompiliatorius praneš apie sintaksės klaidą.

Ką būtina žinoti:

- Programoms sudėtingėjant tikslinga jas padalinti į nedideles lengvai valdomas dalis - funkcijas. Kiekviena funkcija turi turėti savo unikalų vardą. Sugalvokite joms vardus, atsižvelgdami į atliekamas užduotis.
- Funkcijos gali grąžinti reikšmes iškvietusioms funkcijoms. Reikia nurodyti grąžinamos reikšmės tipą prieš funkcijos vardą, priešingu atveju reikia naudoti žodį *void* .
- Programos perduoda informaciją funkcijoms parametrų pagalba. Jei funkcija gauna parametrus, kiekvienam iš jų reikia nurodyti unikalų vardą ir tipą. Jei funkcija negauna parametrų, reikia po funkcijos vardo apvaliuse skliaustuose patalpinti raktinį žodį *void*.
- C++ turi žinoti grąžinamos reikšmės tipą ir parametrų, kuriuos gauna funkcija, tipus. Jei funkcijos nustatymas seka po funkcijos panaudojimo, pradinio failo pradžioje reikia patalpinti funkcijos prototipą.

9 pamoka

Parametrų reikšmių keitimas

Pamokai baigiantis įsisąvinsite šiuos esminius dalykus:

- Jei funkcija nenaudoja rodyklių arba nuorodų, ji negali keisti parametrų reikšmių.
- Parametro reikšmės keitimui funkcija turi žinoti parametro adresą atmintyje.
- Adreso operatorius C++ (&) leidžia programai nustatyti kintamojo adresą atmintyje.
- Kai programa sužino atminties adresą, ji gali naudoti C++ operaciją (*) reikšmės, saugomos duotu adresu, nustatymui.
- Jei programai reikia reikia pakeisti funcijos parametrų reikšmes, ji perduoda funkcijai parametro adresą.

Programa NOCHANGE.CPP perduoda du parametrus, kurių vardai *big* ir *small* funkcijai *display_values*. Pastaroji priskiria abiems parametrams skaičių 1001 ir po to išveda kiekvieno parametro reikšmę. Kai funkcija baigiama, programa atsinaujina ir išveda šių parametrų reikšmes:

```
#include <iostream.h>
void display_values(int a, int b)
{
```

```
  a = 1001;
  b = 1001;

 cout << " Reikšmės funkcijoje  display_values lygios"
       << a << "ir " << b << end1;
}
void main(void)
{
  int big = 2002, small = 0 ;

 cout << "Reik6m4 iki funkcijos " << big << " ir " <<
 small << end1;

display_values(big, small) ;

cout << "Reikšmės po funkcijos " << big << " ir " <<
small << end1 ;
}
```

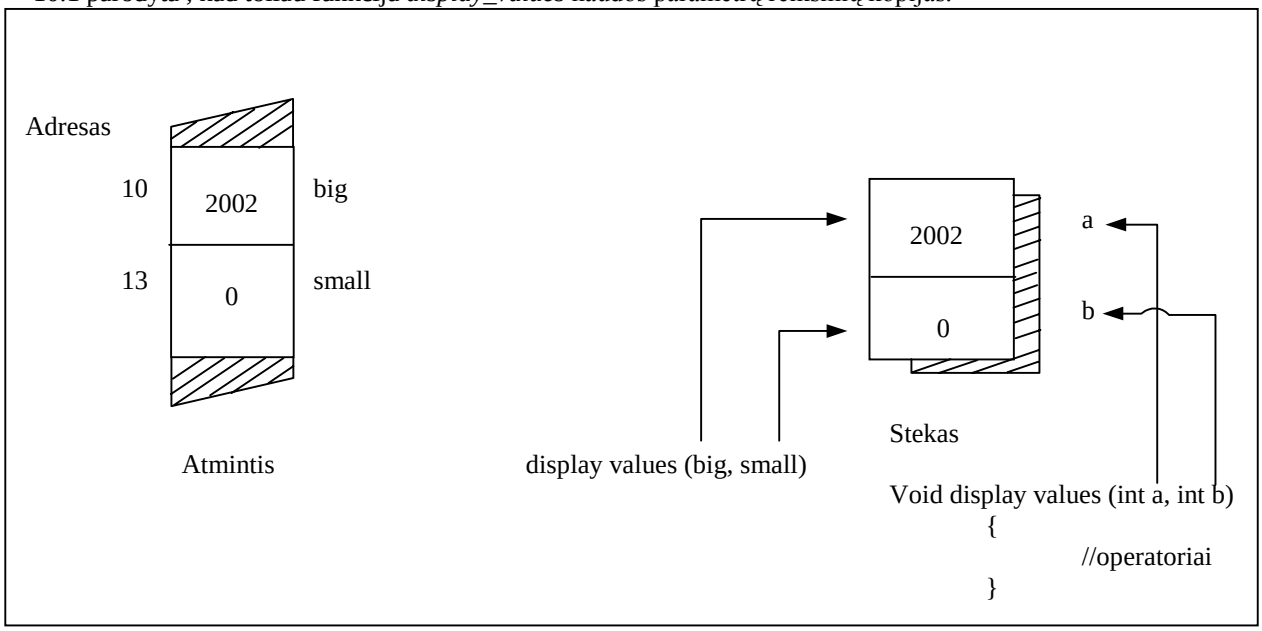
Kai sukompiliuositeir paleisite šią programą, ekrane pamatysite:

C:\ NOCHANGE <ENTER>
Reikšmės iki funkcijos 2002 ir 0
Reikšmės funkcijoje display_values lygios 1001 ir 1001
Reikšmės po funkcijos 2002 ir 0

Kaip matote parametrų reikšmės funkcijoje *display_values* buvo pakeistos (1001). Tačiau grįžus iš funkcijos kintamųjų *big* ir *small* reišmės išliko nepakitę. Norint suprasti, kodėl prametrų pokyčiai nepaveikė kintamųjų *big* ir *small* reikia suprasti kaip C++ perduoda parametrus į funkcijas.

Kai programa perduoda funkcijai parametrą, tai pagal nutyėjimą C++ daro parametro reikšmės kopiją ir patalpina ją į dėklą. Po to funkcija naudoja reikšmės kopiją savo operacijoms. Funkcijai pasibaigus, C++ pašalina dėklo turinį ir visus pokyčius, kurius funkcija padarė parametro reikšmės kopijoje.

Kaip žinote, kintamasis turi vardą, programos priskiriamą atminties ląstelei, kurioje saugoma tam tikro tipo reikšmė. Tarkime, kad kintamieji *big* ir *small* yra 10-oje ir 12-oje atminties ląstelėse. Jei perduosite kintamuosius funkcijai *display_values* , C++ patalpins šių kintamųjų kopijas į dėklą. Pav. 10.1 parodyta , kad toliau funkcija *display_values* naudos parametrų reikšmių kopijas.



Pav. 10.1. C++ parametų kopijas patalpina laikinoje atminties dalyje, vadinamoje dėklu.

Funkcija *display_values* gali kreiptis į steko turinį, kuriame yra reikšmių 2002 ir 0 kopijos. Kadangi *display*_values nieko nežino apie atminties dalis *big* ir *small* (adresai 10 ir 12), ji negali pakeisti realių kintamųjų reikšmių.

Tam kad pakeisti parametrų reikšmes funkcija turi žinoti to parametro adresą. Tam kad pasakyti funkcijai parametro adresą, programos turi naudoti adreso operatorių (&).

Sekantis funkcijos iškvietimas iliustruoja tai, kaip programa panaudos adreso operatorių kintamųjų *big* ir *small* perdavimui į funkciją change_values:

change_values (big, &small); parametrų perdavimas pagal adresą

Funkcijos viduje turite pranešti C++, kad programa perdavinės parametrus pagal adresą. Tam jūs parodote kintamuosius–nuorodas, papildant kiekvieną vardą žvaigždute, kaip parodyta žemiau:

*void change_values (int *big, int *small);* nuoroda, kad int tipo

Nuoroda–kintamasis, tai kintamasis, turintis atminties adresą. Funkcijos viduje turite pranešti C++, kad funkcija dirba su parametro adresu. Tam parametro vardą reikia pažymėti žvaigždute, kaip parodyta žemiau:

```
*big = 1001;
*small = 1001;
```

Programa CHGPARAM.CPP adreso operatorių naudoja parametrų *big* ir *small* adresų perdavimui į funkciją *change_values.* Funkcija panaudoja nuorodas į parametrų atminties ląsteles. Taigi funkcijos padaryti parametrų pakeitimai lieka ir jai pasibaigus:

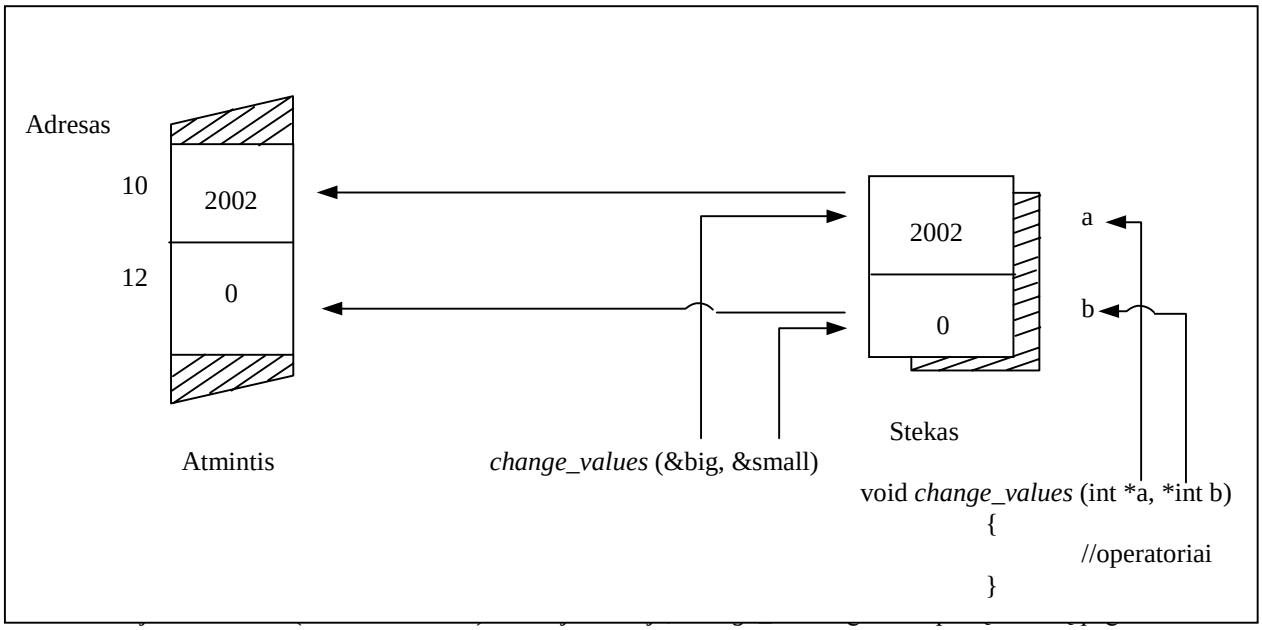
```
#include <iostream.h>

#include <iostream.h>

void change_values (int *a, int *b)
{
        *a = 1001;
        *b = 1001;

        cout <<"reikšmės funkcijoje display_values
        << " lygios" << *a << "ir" << *b << end1;
}

void main (void)
{
        int big = 2002, small = 0"

        cout << "Reikšmės prieš funkciją" << big <<"ir"<< small << end;

        change_values (&big, &small);

        cout <<"reikšmės po funkcijos"<< big << "ir" << small << end1;
}
```

Kai sukompiliuosite ir paleisite programą, ekrane atsiras šis tekstas:

```
C:\> CHGPARAM <ENTER>
Reikšmės prieš funkciją 2002 ir 0
Reikšmės funkcijoje display_values lygios 1001 ir 1001
```

Reikšmės po funkcijos 1001 ir 1001

Kaip matote, reikšmės, kurias funkcija *change_values* supranta kaip parametrus, lieka ir po funkcijos užbaigimo. Kad suprasti, kodėl pakeitimai, atlikti funkcijos kintamiesiems, liko jai pasibaigus, būtina atsiminti, kad funkcija turi priėjimą prie kiekvienos kintamojo atminties ląstelės. Jeigu perduodate parametrus pagal adresą, C++ kiekvieno kintamojo adresą padės į dėklą, kaip parodyta pav. 10.2.



kiekvieno parametro adresą keisdama parametrų reikšmes.

Tam kad pakeisti parametro reikšmę, funkcija turi žinoti to parametro adresą atmintyje. Programa turi perduoti parametro adresą, C++ operatoriaus pagalba:

Some_function (&some_variable);

Funkcijos viduje jūs turite pranešti C++, kad funkcija dirbs su atminties adresu (nuoroda). Tai atliekama su žvaigždute:

*Void some_function (int *some_variable);*

Toliau, funkcijos viduje, jūs turite naudoti žvaigždutę prieš kintamojo vardą:

**some_variable = 1001;*

*cout << *some_variable;*

Kad išvengti klaidų C++ neleis jūsų programai perduoti kintamojo adresą į funkciją, kuri nelaukia nuodoros kaip parametro. Be to, C++ paprastai generuoja kompiliatoriaus perspėjimą, kai programa bando perduoti reikšmę į funkciją, kuri laukia nuorodos kaip parametro.

Jei programa perduoda nuorodas į parametrus, jie gali būti bet kokio tipo. Funkcija, kuri naudoja nuorodas, parodo kintamuosius to paties tipo, pažymėdama kiekvieno kintamojo vardą žvaigždute, patvirtinančia, kad toks kintamasis yra nuoroda. Programa SWAPVALS.CPP perduoda dviejų *float* tipo parametrų adresus į funkciją *swap_values.* Funkcija, savo ruožtu, naudoja nuorodas kiekvienam parametrui, tam kad sukeisti jų reikšmes:

```
#include <iostream.h>

void swap_values (float *a, float *b)
{
        float temp;
```

```
        temp = *a;
        *a = *b;
        *b = temp;
}

void main (void)
{
        float big = 10000.0;
        float small = 0.00001;

        swap_values (&big, &small);

        cout << "big turi" << big << end1;
        cout << "small turi" << small << end1;
}
```

Kaip matote, funkcija parodo *a* ir *b* kaip *float* tipo reikšmių nuorodas:

*Void swap_values (float *a, float *b)*

Tačiau funkcija kintamąjį *temp* nurodo kaip *float,* o ne kaip nuorodą į *float*:

Float temp;

Išnagrinėkime kitą operatorių:

*temp = *a;*

Šis operatorius liepia C++ kintamąjam *temp* priskirti reikšmę, nurodomą kintamuoju *a* (t. y. *big* reikšmė lygi 10000.0). Kadangi *temp* turi *float* tipą, priskyrimas vyksta korektiškai. Kintamasis–nuoroda yra kintamasis, kuris saugo adresą. Sekanti C++ eilutė sako, kad *temp* yra rodyklė į atminties ląstelę, turinčią tipo *float* reikšmę:

*float *temp;*

Čia *temp* gali saugoti reikšmės adresą su plaukiojančiu kableliu, bet ne pačią reikšmę.

Geriausias būdas suprasti, kaip C++ kompiliatorius interpretuoja nuorodas, yra asemblerinio programos teksto nagrinėjimas. Skaitant asemblerio listingą jums bus lengviau suprasti kaip kompiliatorius naudoja dėklą kai perduoda parametrus į funkciją.

Ką būtina žinoti:

- Kai programa perduoda parametrą į funkciją, C++ patalpina parametro reikšmės kopiją į laikinąją atminties dalį, vadinamą dėklu. Bet kokie pakeitimai, kuriuos funkcija vykdo parametrui, vykdomi tik šiai kopijai.

- Tam kad pakeisti parametro reikšmę, funkcija turi žinoti jo adresą.

- Naudodami adreso operatorių (&), jūs galite perduoti kintamojo adresą į funkciją.

- Kai funkcija gauna kintamojo adresą, ji kintamąjį traktuoja kaip rodyklę (kintamojo vardą pažymėti žvaigždute).

- Jeigu funkcijai reikia panaudoti reikšmę, kuri yra nurodyta su rodykle, ji turi pažymėti kintamojo–nuorodos vardą žvaigždute.

10 Pamoka

Run Time bibliotekų (RTB) panaudojimas

Žinoma, kad prieš kviečiant funkciją, C++ kompiliatorius turi sužinoti funkcijos prototipą arba jos apibrėžimą. Kadangi RTB funkcijos programoje neapibrėžtos , reikia nurodyti prototipą kiekvienai funkcijų bibliotekai, kurią žadate naudoti. Kad supaprastinti funkcijų bibliotekų panaudojimą C++ kompiliatorius turi specialius failus, turinčius korektiškus prototipus (jie dar vadinami header-io failais). Taigi į programas reikia įtraukti reikalingą failą operatoriaus *#include* pagalba, o vėliau naudoti reikalingą funkciją.

Pavyzdžiui programa SHOWTIME.CPP naudos RTB funkcijas *time* ir *ctime*, tam kad išvesti datą ir laiką. Šios dvi RTB funkcijos yra faile *teime.h*:

```
#include <iostream.h>
#include <time.h>                    //vykdymo etapo bibliotekos funkcijai

void main (void)
        {
                time_t system_time
                system_time = time (NULL);
                cout << "Einantis sisteminis laikas" <<
                        ctime (&system_time) << end1;
        }
```

Kai jūs sukompiliuosite ir paleisite šią programą, jūsų ekrane atsiras sisteminis laikas ir data:

```
C:\> CHOWTIME <ENTER>
Einantis sisteminis laikas Mon Jan 01 16:13:51 2000
```

Kaip matote, programa naudoja funkcijas *time* ir *ctime.* Funkcijos *ctime* atveju programa perduoda kintamojo *system_time* adresą, naudodama adreso operatorių, aprašytą 9 pamokoje. Šių funkcijų panaudojimui reikia tiesiog prijungti failą *time.h* į pagrindinio failo pradžią.

Panašiu būdu programa SQRT.CPP naudoja funkciją *sqrt*, kelių reikšmių kvadratinės šaknies skaičiavimui. Funkcijos *sqrt* prototipas yra faile *math.h*:

```
#include <iostream.h>
#include <math.h>                    // Turi sqrt prototipą

void main (void)
        {
                cout    << "100.0 kvadratinė šaknis lygi"
                        << sqrt (100.00) << end1;
                cout    << "10.0 kvadratinė šaknis lygi"
                        << sqrt (10.0) << end1;
                cout    << "5,0 kvadratinė šaknis lygi" << sqrt (5.0) << end1;
```

Programa SYSCALL.CPP naudoja funkciją *system*, kurios prototipas apibrėžiamas faile *stdlib.h.* Funkcija *system* leidžia realizuoti operacinės sistemos komandas, tokias kaip "DIR":

```
#include <stdlib.h>

void main (vois)
        {
                system ("DIR");
        }
```

Šiuo atveju programa naudoja funkciją *system* komandos MS–DOS DIR iškvietimui. Raskite laiko eksperimentams su šia programa, paleisdami kitas komandas ar netgi vieną iš jūsų sukurtų programų, išmokus šią knygą.

C++ kompiliatorius turi šimtus RTB funkcijų. Su kompiliatoriumi pateikiama dokumentacija turi pilną visų RTB funkcijų aprašymą. Jeigu ją peržiūrėsite, tai pamatysite, kad paprastai funkcijos naudoja paprastus prototipus. Pavyzdžiui, funkcijai *sqrt* jūs galėtumėte rasti tokį prototipą:

double sqrt(double);

Duotu atveju funkcijos prototipas jums praneša, kad funkcija grąžina *double* tipo reikšmę ir laukia taip pat *double* tipo parametro.

Galima rasti ir sekantį prototipą funkcijai *time*:

*Time_t time (time_t *);*

Prototipas praneša, kad funkcija grąžina *time_t* tipo reikšmę (šis tipas apibūdintas failu *file.h).* Funkcija mano, kad parametras turi būti su nuoroda į *time_t* tipo kintamąjį. Skaitydami funkcijų dokumentaciją, jūs labai daug sužinosite apie pačias C++ funkcijas, taip pat atkreipkite dėmesį į funkcijų prototipus.

Kitas būdas įsisavinti kompiliatoriaus RTB funkcijas yra peržiūrėti *.h failus, esančiuse pakatologyje INCLUDE.

Standartinių RTB papildymui kompiliatoriai dar turi ir API funkcijas arba papildomų programų sąsają. Jeigu programuojate Windows terpėje, tai jums reiks grafinių API funkcijų, telefoninės API (TAPI), API multimedijai ir t. t. Prieš sukuriant savo nuosavas funkcijas, įsitikinkite, kad neradote tokių API funkcijų, kurios pridedamos prie kompiliatoriaus.

Ką jums būtina žinoti:

C++ RTB turi didelį kiekį funkcijų, kurias galima naudoti programose. Nepagailėkite laiko, vykdymo etapo bibliotekų dokumentacijos nagrinėjimui. Išsiaiškinkite RTB funkcijų paskirtį. Šių funkcijų privalumas tas, kad jas naudodami sutaupysite daug laiko programuodami.

- Norint panaudoti RTB funkcijas reikia nurodyti jos prototipą;

- Daugelis C++ kompiliatorių turi *.h failus, kurie turi korektiškus prototipus kiekvienai funkcijų bibliotekai.

- RTB papildymui daugelis C++ kompiliatorių turi API funkcijas (papildomų programų sąsają) konkrečių uždavinių vykdymui, pavyzdžiui grafikos programavimui arba multimedijai.

11 Pamoka

Lokaliniai ir globaliniai kintamieji

Kaip žinote, funkcijos leidžia suskaidyti programą į nedideles, lengvai valdomas dalis. Kintamieji, naudojami funkcijos viduje, vadinami lokaliniais. Jų reikšmė ir faktas, kad jie egzistuoja yra žinomi tik toje funkcijoje. Pavyzdžiui, jei apsirašote lokalinį kintamąjį salary funkcijoje *payroll*, tai kitos funkcijos negali gauti kintamojo *salary* reikšmės. Kitoms funkcijoms šis kintamasis net neegzistuoja. Šioje pamokoje nagrinėjamas kintamojo galiojimo laukas arba programos dalis, kurioje šis lokalinis kintamasis egzistuoja. Šios pamokos pabaigoje susipažinsite su pagrindinėmis koncepcijomis:

- lokaliniai kintamieji funkcijos viduje aprašomi taip pat, kaip ir pagrindinėje programoje *main*, t.y. nurodant kintamojo tipą ir vardą.
- Kintamųjų, naudojamų funkcijos viduje, vardai turi būti unikalūs tik konkrečioje funkcijoje.
- Kintamojo galiojimo laukas apibrėžia programos dalį, kur kintamasis yra žinomas ir prieinamas
- Globaliniai kintamieji skirtingai negu kad lokaliniai yra žinomi visoje programoje ir prieinami iš visų funkcijų.
- Globalinio lauko galiojimo operatorius C++ (::) leidžia valdyti kintamojo galiojimo lauką.

Kintamasis vadinamas lokaliniu, nes egzistuoja tik atatinkamos funkcijos viduje. Lokalinius kintamieji aprašomi funkcijos pradžioje, po figūrinių skliaustų:

```
void some_function(void)
  {
      int count;
      float result;
  }
```

Programa USEBEEPS.CPP naudoja funkciją *sound_speaker*, kuri priverčia kompiuterio garsiakalbį pypsėti tiek kartų, kiek nurodyta parametre *beep*. Funkcijoje *sound_speaker* lokalinis kintamasis *counter* saugo garsų, girdimų per garsiakalbį, skaičių:

```
#include <iostream.h>

void sound_beeps(int beeps)
 {
  for (int counter = 1; counter <= beeps; counter++)
    cout << '\a';
 }

void main(void)
 {
  sound_beeps(2);
  sound_beeps(3);
 }
```

Kaip matome, funkcija *sound_beeps* aprašo kintamąjį *counter* iš karto po figūrinių skliaustų. Kadangi *counter* aprašomas funkcijos *sound_beep*s viduje , šis kintamasis yra lokalinis .

Aprašant lokalinį kintamąjį vienos funkcijos viduje, dažnai tokio pat vardo kitas lokalinis kintamasis gali būti aprašytas kitos funkcijos viduje. Tačiau jei dvi funkcijos naudoja lokalinius kintamuosius tokiais pat vardais, tai nesukelia konflikto. C++ traktuoja kintamojo vardą kaip lokalinį konkrečioje funkcijoje.

Programa LCLNAME.CPP naudoja funkciją *add_values* dviejų sveikų skaičių sudėčiai. Ši funkcija rezultatą priskiria lokaliniam kintamajam *value*. Taip pat ir *main* vienas iš parametrų, perduodamų funkcijai, turi tokį pat vardą *value*. Kadangi C++ abu kintamuosius traktuoja kaip lokalinius konkrečioms funkcijoms, jų vardai nekonfliktuoja:

```
#include <iostream.h>

int add_values(int a, int b)
  {
      int value;

      value = a + b;

      return(value);
  }

void main(void)
 {
   int value = 1001;
   int other_value = 2002;

   cout << value << " + " << other_value << " = " <<
            add_values(value, other_value) << endl;
}
```

Programose C++ leidžia naudoti ir **globalinius** kintamuosius, kurie žinomi visai programai (globalūs visoms funkcijoms). Globalinius kintamuosius reikia aprašyti programos pradžioje (ne kurioje nors funkcijoje):

```
int some_global_variable;

void main(void)
  {
      //čia turi būti programos operatoriai
  }
```

Programa GLOBAL.CPP naudoja globalinį kintamąjį *number*. Kiekviena funkcija programoje gali naudoti (arba pakeisti) globalinio kintamojo reikšmę. Duotam pavyzdyje kiekviena funkcija išveda momentinę reikšmę, o paskui ją padidina vienetu:

```
#include <iostream.h>

int number = 1001;

void first_change(void)
  {
cout << ”number reiksme first_change “<< number << endl;
      number++;
  }

void second_change(void)
  {
cout << ”number reiksme second_change “<<number << endl;
      number++;
  }

void main(void)
  {
cout << ”number reiksme main “<< number << endl;
      number++;
      first_change();
      second_change();
  }
```

Patartina vengti naudoti globalinius kintamuosius programose. Kadangi kiekviena funkcija gali pakeisti globalinio kintamojo reikšmę, sunku atsekti visas funkcijas, kurios galėjo pakeisti kintamojo reikšmę. Vietoj to verta aprašyti kintamąjį viduje *main*, o paskui perdavinėti funkcijoms kaip parametrą.

Jei vis dėlto jums reikia naudoti globalinius kintamuosius, gali konfliktuoti jų vardai. Iškilus tokiam konfliktui C++ prioritetą teikia lokaliniam kintamajam.

Gali būti situacijų, kai jums reiks kreiptis į globalinį kintamąjį, kurio vardas konfliktuoja su lokalinio kintamojo vardu. Tokiu atveju programa gali naudoti *globalinį* C++ s*prendimų operatorių* (::). Tarkime pas jus yra lokalinis ir globalinis kintamasis, kurių vardai *number*. Jei funkcija, nori naudoti lokalinį kintamąjį *number*, ji paprasčiausiai kreipiasi į kintamąjį kaip parodyta:

```
number = 1001;
```

Jei funkcija nori kreiptis į globalinį kintamąjį, programa naudoja globalinį sprendimų operatorių kaip parodyta:

```
::number = 2002;
```

Programa GLOBLOCA.CPP naudoja globalinį kintamąjį *number*. Funkcija *show_numbers* naudoja lokalinį kintamąjį *number*. Ši funkcija naudoja globalinį sprendimų operatorių kreipimuisi į globalinį kintamąjį:

```
#include <iostream.h>
```

```
int number = 1001;

void show_numbers(int number)
  {
cout << “Lokalinis kintamasis number” << “lygus” << number << endl;
cout << “Globalinis kintamasis number” << “lygus” << ::number << endl;
  }

void main(void)
 {
        int some_value = 2002;

        show_numbers(some_value);
 }
```

Sukompiliavus ir paleidus programą, ekrane bus toks vaizdas:

C:\> GLOBLOCA <ENTER>
Lokalinis kintamasis number lygus 2002
Globalinis kintamasis number lygus 1001

Skaitant knygas ar straipsnius apie C++ galima sutikti tokį terminą, kaip *galiojimo laukas*. Jis parodo programos dalį, kurioje kintamojo vardas turi prasmę ir yra vartojamas. Lokalinio kintamojo galiojimo laukas yra funkcijos, kurioje jis yra aprašytas, viduje. Globaliniai kintamieji žinomi visoje programoje, todėl jų galiojimo laukas yra didelis.

12 Pamoka

Funkcijų perkrovimas

Aprašant funkciją, programoje turite nurodyti rezultato tipą, o taip pat perduodamų parametrų skaičių ir jų tipą. Jei anksčiau (programuojant su C) jūs turėjote funkciją *add_values*, kuri dirbo su dviem sveikais skaičiais, o jums reikia funkcijos, kuri sudėtų tris sveikus skaičius, reikėjo sukurti naują funkciją su kitu pavadinimu. Pavyzdžiui, galėjote naudoti funkcijas *add_two_values* ir *add_three_values*. Analogiškai, jeigu jums reikėjo panašios funkcijos *float* tipo kintamųjų sudėčiai. Tam taip pat reikėjo naujos funkcijos. Norint išvengti funkcijų dubliavimo, C++ leidžia apsirašyti keletą funkcijų tuo pačiu vardu. C++ kompiliatorius tokiu atveju atkreipia dėmesį į kiekvienos funkcijos argumentų kiekį ir tuomet iškviečia reikiamą funkciją. Kompiliatoriaus pasirinkimas tarp kelių funkcijų, vadinamas *perkrovimu*. Šioje pamokoje išmoksite naudoti perkrautas funkcijas, o pabaigoje susipažinsite su tokiomis pagrindinėmis koncepcijomis:

- Funkcijų perkrovimas leidžia naudoti tą patį vardą kelioms funkcijoms su skirtingo tipo parametrais.
- Funkcijų perkrovimui paprasčiausiai apsirašykite dvi funkcijas tuo pačiu vardu ir rezultato tipu, kurios skirtųsi parametrų skaičiumi ar tipu.

Funkcijų perkrovimas yra išskirtinis C++ bruožas. Tai labai patogu ir programa tampa aiškesnė.

Funkcijų perkrovimas leidžia programai naudoti kelias to paties vardo ir rezultato tipo funkcijas. Sekanti programa perkrauna funkciją vardu *add_values*. Pirmoji funkcija sudeda du *int* tipo skaičius. Antroji – sudeda tris skaičius. Kompiliacijos metu C++ kompiliatorius išsirenka reikiamą funkciją:

```
#include <iostream.h>

int add_values(int a, int b)
 {
        return(a + b);
 }

int add_values(int a, int b, int c)
 {
```

```
        return(a + b + c);
 }

void main(void)
 {
        cout << “200 + 801 = “ << add_values(200, 801) << endl;
        cout << “100 + 201 + 700 = “ << add_values(100, 201, 700) << endl;
 }
```

Kaip matote, programa naudoja dvi funkcijas vardu *add_values*. Pirmoji funkcija sudeda du *int* tipo skaičius, o antroji – tris skaičius. Kompiliatoriui nereikia kokiu nors būdu pranešti apie funkcijų perkrovimą. Kompiliatorius pats atpažįsta, kurią funkciją naudoti su duotais parametrais.

Panašiai programa MSG_OVR.CPP perkrauna funkciją *show_message*. Pirmoji funkcija *show_message* išveda standartinį pranešimą ir jai parametrai neperduodami. Antroji išveda perduodamą šiai funkcijai pranešimą, o trečioji išveda du pranešimus:

```
#include <iostream.h>

void show_message(void)
 {
        cout << “Standartinis pranesimas: “
                << “Mokomes programuoti C++” << endl;
 }

void show_message(char message)
 {
        cout << message << endl;
 }

void show_message(char *first, char *second)
 {
        cout << first << endl;
        cout << second << endl;
 }

void main(void)
 {
        show_message();
        show_message( “Mokomes programuoti C++!”);
        show_message(“C++ nera neaiškumu!”,
                        “Perkrovimas – puikus dalykas!”);
 }
```

Funkcijų perkrovimas supaprastina programų skaitymą. Jis leidžia programai naudoti kelias funkcijas su tuo pačiu vardu. Perkrautos funkcijos turi grąžinti vienodo tipo rezultatus[*], bet gali skirtis parametrų skaičiumi ir tipu.

[*] Perkrautos funkcijos neprivalo gražinti vienodo tipo rezultatų, nes kompiliatorius vienareikšmiškai identifikuoja funkciją pagal jos vardą ir argumentus. Kompiliatoriui funkcijos su vienodais vardais, bet skirtingais argumentų tipais yra skirtingos funkcijos.

Funkcijų perkrovimas leidžia nurodyti keletą paskirčių tai pačiai funkcijai. Kompiliavimo metu C++ nuspręs, kokią funkciją reikia naudoti, atsižvelgiant į reikalingų perduoti parametrų kiekį ir tipą. Prieš pereidami prie sekančios pamokos, įsitikinkite, kad išmokote šias pagrindines koncepcijas:

- Funkcijų perkrovimas suteikia keletą "požiūrių" į vieną ir tą pačią funkciją programoje.
- Funkcijos perkrovimui tiesiog aprašykite keletą funkcijų su tuo pačiu vardu ir grįžtamosios reikšmės tipu, kurios skiriasi tik parametrų kiekiu ir tipu.
- Kompiliuodama C++ nuspręs, kokią funkciją iškviesti pagal parametrų kiekį ir tipą.
- Funkcijų perkrovimas supaprastina programavimą, leisdamas programuotojams dirbti su vienu funkcijos vardu.

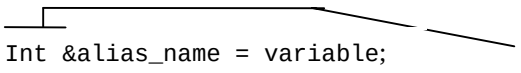
13 Pamoka

Persiuntimų panaudojimas

9-toje pamokoje sužinojote, kaip keisti keisti parametrus funkcijų viduje rodyklių pagalba. Nuorodų panaudojimui jūs turite pažymėti kintamųjų-rodyklių vardus žvaigždute. Rodyklių panaudojimas buvo "paveldėtas" iš C kalbos. Parametrų pakeitimo proceso supaprastinimui C++ įveda sąvoką *persiuntimas*. Šioje pamokoje jūs sužinosite, kad persiuntimas – tai pseudonimas (arba antras vardas), kurį programos gali naudoti kreipimuisi į kintamąjį. Iki pamokos pabaigos sužinosite šias pagrindines koncepcijas:

- Persiuntimo deklaravimui ir inicializacijai programoje aprašykite kintamąjį, užrašydami ampersandą (&) iškart po kintamojo tipo, o po to naudokite priskyrimo operatorių pseudonimo paskirčiai apibrėžti, pavyzdžiui: *int &alias_name=variable;*.
- Jūsų programos gali perdavinėti persiuntimus į funkcijas kaip parametrus, o funkcija savo ruožtu gali keisti atitinkamas parametro reikšmes, nenaudodama nuorodų.
- Funkcijos viduje jūs turite deklaruoti parametrą kaip persiuntimą, pridedant ampersando ženklą (&) po parametro tipo, po to galima keisti parametro reikšmę funkcijos viduje nenaudojant nuorodų.

Persiuntimas C++ leidžia sukurti sinonimą (arba antrą vardą) programos kintamiesiems. Persiuntimo deklaravimui programos viduje nurodykite ampersando ženklą (&) betarpiškai po parametro tipo. Deklaruodami persiuntimą jūs turite tuojau pat priskirti jai kintamąjį, kuriam šis persiuntimas bus sinonimu taip, kaip parodyta žemiau:



```
Int &alias_name = variable;
```

Persiuntimo deklaravimas

Po persiuntimo deklaravimo jūsų programa gali naudoti kaip kintamąjį, taip ir persiuntimą:

```
alias_name = 1001;
variable = 1001;
```

Programa SHOW_REF.CPP sukuria persiuntimą su vardu *alias_name* ir priskiria sinonimui kintamąjį *number*. Toliau programa naudoja kaip persiuntimą, taip ir kintamąjį:

```
#include <iostream.h>

void main(void)
  {
    int number = 501;
    int &alias_name = number;  //Sukurti persiuntim•

    cout <<"Kintamojo number reikšm•" << number << endl;
```

```
    cout <<"number pseudonimo reikšm•" << alias_name <<endl;

    alias_name = alias_name + 500;

    cout <<"Kintamojo number reikšm•" << number << endl;
    cout <<"number pseudonimo reikšm•" << alias_name <<endl;
   }
```

Kaip matote, programa prideda 500 prie persiuntimo *alias_name*. Taipogi programa prideda 500 ir prie atitinkamo kintamojo *number*, kuriam persiuntimas tarnauja sinonimu arba antru vardu. Sukompiliavus ir paleidus šią programą, į ekraną bus išvesta:

```
C:\> SHOW_REF <ENTER>
Kintamojo number reikšm• 501
number pseudonimo reikšm• 501
Kintamojo number reikšm• 1001
number pseudonimo reikšm• 1001
```

Persiuntimo deklaravimas

Persiuntimas yra sinonimas (antras vardas), kurį programos gali naudoti kreipimuisi į kintamąjį. Persiuntimo deklaravimui nurodykite ampersando ženklą (&) betarpiškai po kintamojo tipo, po kurio seka lygybės ženklas ir vardas kintamojo, kuriam persiuntimas tarnauja pseudonimu.

```
Float &salary_alias = salary;
```

Pagrindinė persiuntimų paskirtis yra parametrų keitimo funkcijos viduje proceso supaprastinimas. Programa REFERENC.CPP priskiria persiuntimą vardu *number_alias* kintamajam *number*. Jiperduoda kintamojo persiuntimą funkcijai *change_value*, kuri priskiria kintamajam reikšmę 1001:

```
#include <iostream.h>

void change_value(int &alias)
  {
    alias = 1001;
  }

void main(void)
  {
    int number;
    int& number_alias = number;

    change_value(number_alias);

    cout <<"Kintamojo number reikšm•" << number <<endl;
   }
```

Kaip matote, programa perduoda persiuntimą funkcijai *change_value*. Jei peržiūrėsite funkcijos deklaravimą, pastebėsite, kad *change_value* aprašo parametrą *alias* kaip persiuntimą su tipu *int*:

```
Void change_value(int &alias)
```

Funkcijos *change_value* viduje galite keisti parametro reikšmę be nuorodų pagalbos. To rezultate nenaudojama žvaigždutė (*) ir operacija funkcijos viduje tampa lengviau suprantama.

Peržiūrėkime kitą pavyzdį. 9-toje pamokoje jūs naudojote tokias funkcijas dviejų reikšmių su slankiu kableliu sukeitimui:

```
void swap_values(float *a, float *b)
  {
    float temp;
    temp = *a;
    *a = *b;
```

```
    *b = temp;
  }
```

Kaip matote, funkcija kombinuoja kintamuosius-rodykles su kintamaisiais-nerodyklėmis. Programa SWAP_REF.CPP naudoja persiuntimus reikšmėms su slankiu kableliu funkcijos supaprastinimui:

```
#include <iostream.h>

void swap_values(float& a, float& b)
 {
  float temp;

  temp = a;
  a = b;
  b = temp;
 }

void main(void)
 {
  float big = 10000.0;
  float small = 0.00001;
  float& big_alias = big;
  float& small_alias = small;

  swap_values(big_alias, small_alias);
  cout << "Big reikšm• " << big << endl;
  cout << "Small reikšm• " << small << endl;
 }
```

Kaip matote, funkciją *swap_values* dabar lengviau suprasti, tačiau programa dabar turi du papildomus vardus (persiuntimus *big_alias* ir *small_alias*), kurių turite nepamiršti.

Persiuntimas nėra kintamasis. Vieną kartą priskyrę persiuntimui reikšmę, jau nebegalite jos pakeisti. Be to, skirtingai nuo nuorodų, jūs negalite su persiuntimais atlikti šių operacijų:

- negalite gauti persiuntimo adreso naudodami C++ adreso operatorių;
- negalite priskirti persiuntimui nuorodos;
- negalite palyginti persiuntimų reikšmių, naudodami C++ palyginimo operatorius;
- negalite atlikti aritmetinių operacijų su persiuntimu, pavyzdžiui pridėti vidurkį;
- negalite pakeisti persiuntimo.

Iš šios pamokos sužinojote, kaip panaudoti persiuntimus C++ sinonimo arba antro kintamajo vardo sukūrimui. Persiuntimų panaudojimas gali supaprastinti funkcijas, keičiančias parametrų reikšmes. Iš 15-tos pamokos sužinosite, kad C++ leidžia nustatyti funkcijos parametrus pagal nutylėjimą. Iškviečiant funkciją programa gali praleisti vieno ar kelių parametrų reikšmes ir funkcija naudos reikšmes pagal nutylėjimą. Bet prieš mokydamiesi sekančią pamoką, įsitikinkite, kad išmokote šias pagrindines koncepcijas:

- Persiuntimas C++ yra kintamojo sinonimas (arba antras jo vardas).
- Persiuntimo deklaravimui nurodykite ampersando ženklą (&) betarpiškai po kintamojo tipo, po kurio seka lygybės ženklas ir vardas kintamojo, kuriam persiuntimas tarnauja sinonimu.
- Jūs turėtumėte prijungti keletą komentarų prieš funkcijas, naudojančias persiuntimus parametrų reikšmėms keisti, kad kad kiti programuotojai, skaitantys jūsų kodą, iškart į tai atkreiptų dėmesį.
- Dažnas persiuntimų naudojimas gali labai apsunkinti programinio kodo supratimą.

14 Pamoka

Parametrų reikšmės pagal nutylėjimą

Kaip jūs jau žinote, C++ leidžia parametrų pagalba perduoti informaciją į funkcijas. Iš 12 pamokos išsiaiškinote, kad C++ taip pat palaiko funkcijų perkrovimą, nustatydama skirtingą parametrų kiekį ir net skirtingų tipų parametrus. Iškviečiant funkcijas galima parametrus ir praleisti. Tokiais atvejais praleistiems parametrams bus panaudotos reikšmės pagal nutylėjimą. Ši pamoka aiškina kaip nustatyti reikšmes funkcijos parametrams pagal nutylėjimą. Šios pamokos pabaigoje jūs įsisavinsite tokias pagrindines koncepcijas:

- C++ leidžia programoms nusakyti parametrų reikšmes pagal nutylėjimą;
- Parametrų reikšmės pagal nutylėjimą nusakomos aprašant funkciją;
- Jei funkcijos iškvietimas praleidžia vieną ar kelis parametrus, C++ naudos reikšmes pagal nutylėjimą;
- Jei funkcijos iškvietimas praleidžia nustatyto parametro reikšmę, turi būti praleistos ir visų sekančių parametrų reikšmės;

Reikšmių parametrams nustatymas pagal nutylėjimą supaprastina galimybę *pakartotinai naudoti* funkcijas (naudoti jas keliomis programomis).

Nurodyti reikšmes funkcijos parametrams pagal nutylėjimą labai lengva., kaip parodyta žemiau:

```
void some_funktion(int size=12, float cost=19.95)
 {
    // Funkcijos operatoriai
 }
```

Reikšmės pagal nutylėjimą

Programa DEFAULTS.CPP priskiria parametrams *a*, *b* ir *c* reikšmes pagal nutylėjimą funkcijos *show_parameters* viduje. Paskui programa keturis kartus iškviečia šią funkciją, iš pradžių iš viso nenurodydama parametrų, paskui nunurodydama tik parametrą *a*, paskui reikšmes *a* ir *b*, ir galiausiai nurodydama visų trijų parametrų reikšmes:

```
#include <iostream.h>

void show_parameters(int a=1, int b=2, int c=3)
 {
  cout << ”a” << a <<”b” << b << “c” << c << endl;
 }

void main(void)
 {
  show_parameters();
  show_parameters(1001);
  show_parameters(1001,2002);
  show_parameters(1001,2002,3003);
 }
```

Kai jūs sukompiliuosite ir paleisite šią programą, į jūsų ekraną bus išvesta:

```
C:\> DEFAULTS <ENTER>
a 1 b 2 c 3
a 1001 b 2 c 3
a 1001 b 2002 c 3
a 1001 b 2002 c 3003
```

Kaip matote, jei reikia, funkcija naudoja parametrus pagal nutylėjimą. Jei programa praleidžia nustatytą funkcijos, parenkančios reikšmes pagal nutylėjimą, parametrą, tai reikia praleisti ir visus sekančius parametrus. Negalima praleisti vidurinio parametro. Aukščiau pavaizduotos programos atveju, jei reikėjo praleisti parametro *b* reikšmę funkcijoje *show_parameters*, programa privalėjo praleisti ir parametrą *c*.

Iš šios pamokos sužinojote, kad C++ leidžia nurodyti funkcijos parametrus pagal nutylėjimą. Jei programa praleidžia vieną ar kelis parametrus, tai funkcijos naudoja reikšmes pagal nutylėjimą. Kitose pamokose, kai jūsų programos pradės naudoti objektinį programavimą, naudosite parametrus pagal nutylėjimą įvairių kintamųjų inicializacijai. Kaip jau žinote, kintamasis leidžia saugoti nustatyto tipo reikšmę (*int, float* ir t.t.). 15-toje pamokoje įsisavinsite, kaip saugoti vieno ir to paties tipo reikšmes *masyve*. Pavyzdžiui, jūsų programa gali saugoti testo taškus 100 studentų arba 50 akcijų kainas. Masyvų pagalba saugoti ir panaudoti tokius duomenis labai lengva. Bet prieš mokydamiesi 15 pamoką įsitikinkite, kad išmokote šias pagrindines koncepcijas:

- Norėdami priskirti funkcijos parametrams reikšmes pagal nutylėjimą, naudokite C++ priskyrimo operatorių iškart deklaruodami funkciją.
- Jei funkcijos iškvietimas praleidžia vieną ar kelis parametrus, C++ naudos reikšmes pagal nutylėjimą.
- Jei programa praleidžia vieną funkcijos parametrą, tai reikia praleisti ir visus sekančius parametrus. Programa negali praleisti vidurinio parametro.
- Nurodydami parametrus pagal nutylėjimą tuo pačiu palengvinate savo funkcijų panaudojimą.