OPERATORIŲ PERKROVIMAS

C++ ir kitose programavimo kalbose naudojami operatoriai pasižymi polimorfizmu. Pvz., operatorius + gali būti taikomas tiek **int**, tiek **float** tipo kintamiesiems, nors sudėties veiksmai procesoriuje iš tikrųjų bus atliekami skirtingai.

Programuotojui taip pat suteikiama galimybė perkrauti operatorius darbui su naujai apibrėžtais duomenų tipais (klasėmis). Operatorius C++ kalboje galima perkrauti tiek globaliai, tiek klasės ribose. Perkraunami operatoriai aprašomi naudojant bazinį žodį **operator**. Pvz., norint perkrauti sudėties operatorių nurodoma **operator+**, o kartu su priskyrimu: **operator+=**

Operatorių veikimo keitimas yra vadinamas perkrovimu arba perdengimu. Bendra operatoriaus perkrovimo (perdengimo) sintaksė :

<tipas> **operator**<operatoriaus_simbolis> (<param_tipas> parametras);

Iš tikrųjų, operatoriai – tai funkcijos, todėl operatorių perkrovimas siejamas su funkcijų perkrovimu. Perkrauti galima beveik visus C++ operatorius, išskyrus: **. .* :: ?: #**

Operatorinė funkcija dažnai apibrėžiama klasės viduje kaip klasės metodas.

Binarinių operatorių perkrovimas.

Pavyzdys 3.14. Operatorių perkrovimas klasėje.

```
#include <iostream.h>
class Taskas
{
        int x, y;
   public:
        Taskas() { x=0, y=0; }
        Taskas(int _x, int _y) { x=_x, y=_y; }
        void GautiKoord(int& _x, int& _y) {  _x=x, _y=y;}
        Taskas operator+(Taskas& ob);
        Taskas operator-(Taskas& ob);
};

Taskas
Taskas :: operator+ ( Taskas&  ob)
```

```
{
      Taskas  tempOb;
       tempOb.x = x + ob.x;
       tempOb.y = y + ob.y;
       return tempOb;
};

Taskas
Taskas :: operator- ( Taskas&  ob)
{
      Taskas  tempOb;
       tempOb.x = x – ob.x;
       tempOb.y = y – ob.y;
       return tempOb;
};

main ()
{
    int x, y;
    Taskas  PtA(10,20), PtB(3,8), PtC;
    PtC = PtA + PtB;                //Perkrauto sudėties operatoriaus iškvietimas
    PtC.GautiKoord(x,y);
    cout << "PtC.x =" << x <<  "  PtC.y =" << y << endl;
    PtC = PtA – PtB;                //Perkrauto atimties operatoriaus iškvietimas
    PtC.GautiKoord(x,y);
    cout << "PtC.x =" << x <<  "  PtC.y =" << y << endl;
    return 0;
 }
```

Šios programos vykdymo rezultatas:

```
PtC.x =13    PtC.y =28
PtC.x =7     PtC.y =12
```

Čia naudojamas parametrų perdavimas pagal adresą (nuoroda), o gražina operatorinės funkcijos klasės *Taskas* objektą. Tai leidžia sudaryti, pvz., tokias sudėtines išraiškas:

```
PtC = PtA + PtB – PtC;
(PtA + PtB).GautiKoord(x,y);
```

Sudėties operatoriaus gražinamas laikinas objektas naudojamas metodo *GautiKoord(x,y)* iškvietimui.

Dabar vietoje antro parametro - nuorodos panaudokime standartinio tipo parametrą – reikšmę.

Pavyzdys 3.15. Operatorių perkrovimas su parametru – reikšmė.

```
#include <iostream.h>
class Taskas
{
        int x, y;
   public:
        Taskas() { x=0, y=0; }
        Taskas(int _x, int _y) { x=_x, y=_y; }
        void GautiKoord(int& _x, int& _y) {  _x=x, _y=y;}
        Taskas operator+(Taskas& ob);
        Taskas operator+(int n);
};
Taskas
Taskas :: operator+ ( Taskas&  ob)
{
      Taskas  tempOb;
       tempOb.x = x + ob.x;
       tempOb.y = y + ob.y;
       return tempOb;
};

Taskas
Taskas :: operator+ (int n)
{
      Taskas  tempOb;
       tempOb.x = x + n;
       tempOb.y = y + n;
       return tempOb;
};

main ()
{
    int x, y;
    Taskas  PtA(10,20), PtB(3,8), PtC;
    PtC = PtA + PtB;                //Perkrauto sudėties operatoriaus iškvietimas
    PtC.GautiKoord(x,y);
    cout << "PtC.x =" << x <<  "  PtC.y =" << y << endl;
    PtC = PtA + 30;                 //Perkrauto sudėties (int) operatoriaus iškvietimas
    PtC.GautiKoord(x,y);
    cout << "(PtA+30).x =" << x <<  "  (PtA+30).y =" << y << endl;
    return 0;
 }
```

Šios programos vykdymo rezultatas:

PtC.x =13 PtC.y =28

(PtA+30).x =40 (PtA+30).y =50

Čia parametras *int n* yra dešinysis sudėties operatoriaus operandas, tad kompiliatorius supranta instrukciją

PtC = PtA + 30;

tačiau nesupranta

PtC = 30 + PtA;

Jei norima, kad perkrauta operacija būtų universali, reikia naudoti draugiškas funkcijas. Kadangi draugiškai funkcijai nėra perduodama rodyklė *this*, jai reikės perduoti abu argumentus – kairįjį ir dešinįjį:

Pavyzdys 3.16. Operatorių perkrovimas naudojant draugišką funkciją.

```
#include <iostream.h>
class Taskas
{
int x, y;
   public:
        Taskas() { x=0, y=0; }
Taskas(int _x, int _y) { x=_x, y=_y; }
void GautiKoord(int& _x, int& _y) {  _x=x, _y=y;}
friend Taskas operator+(Taskas& ob1, Taskas& ob2);
friend Taskas operator+( Taskas& ob, int n);
friend Taskas operator+( int n, Taskas& ob);
};

// Objektų sudėties operatoriaus perkrovimas
Taskas operator+(Taskas& ob1, Taskas& ob2)
{
 Taskas  tempOb;
 tempOb.x = ob1.x + ob2.x;
 tempOb.y = ob1.y + ob2.y;
 return tempOb;
};

// Sudėties operatoriaus  obj + int  perkrovimas
Taskas operator+( Taskas& ob, int n)
{
  Taskas  tempOb;
```

```
  tempOb.x = ob.x + n;
  tempOb.y = ob.y + n;
  return tempOb;
};

// Sudėties operatoriaus  int + obj  perkrovimas
Taskas operator+(int n, Taskas& ob)
{
  Taskas  tempOb;
  tempOb.x = n + ob.x;
  tempOb.y = n + ob.y;
  return tempOb;
};
main ()
{
    int x, y;
    Taskas  PtA(10,20), PtB(3,8), PtC;
    PtC = PtA + PtB;              //Perkrauto sudėties operatoriaus iškvietimas
    PtC.GautiKoord(x,y);
    cout << "PtC.x =" << x <<  "  PtC.y =" << y << endl;
    PtC = PtA + 30;               //Perkrauto operatoriaus obj+int iškvietimas
    PtC.GautiKoord(x,y);
    cout << "(PtA+30).x =" << x <<  "  (PtA+30).y =" << y << endl;
    PtC = 30 + PtA;               //Perkrauto operatoriaus int+obj iškvietimas
    PtC.GautiKoord(x,y);
    cout << "(30+PtA).x =" << x <<  "  (30+PtA).y =" << y << endl;

    return 0;
 }
```

Šios programos vykdymo rezultatas:

PtC.x =13 PtC.y =28

(PtA+30).x =40 (PtA+30).y =50

(30+PtA).x =40 (30+PtA).y =50

Tokiu būdu draugiškos funkcijos leido įgyvendinti visus perkrovimo variantus ir dabar abi instrukcijos

PtC = PtA + 30

ir

PtC = 30 + PtA

yra teisingos.

Santykio ir loginių operatorių perkrovimas.

Tai binarinės operacijos, tačiau perkraunant jas, operatorinė funkcija gražina ne klasės objektą, o logines reikšmes (arba *int* tipo reikšmės, traktuojamos kaip loginės).

Pavyzdys 3.17. Santykio ir loginių operatorių perkrovimas.

```
#include <iostream.h>
class Taskas
{
        int x, y;
   public:
        Taskas() { x=0, y=0; }
        Taskas(int _x, int _y) { x=_x, y=_y; }
        void GautiKoord(int& _x, int& _y) {  _x=x, _y=y;}
        bool operator==(Taskas ob);
        bool operator&&(Taskas ob);
};
bool
Taskas :: operator== ( Taskas  ob)
{
      return (x == ob.x && y == ob.y);
};
bool
Taskas :: operator&& ( Taskas  ob)
{
      return (x && ob.x) && (y && ob.y);
};
main ()
{
    int x, y;
    Taskas  Pt1(10,20), Pt2(10,25), Pt3(10,20), Pt4;

    if  (Pt1== Pt2) cout << "Pt1 = Pt2\n";
    else cout << "Pt1 ≠ Pt2\n";

    if  (Pt1== Pt3) cout << "Pt1 = Pt3\n";
    else cout << "Pt1 ≠ Pt3\n";

    if  (Pt1 && Pt3) cout << "Pt1 && Pt3 yra teisinga\n";
    else cout << "Pt1 && Pt3 yra neteisinga \n";

    if  (Pt1 && Pt4) cout << "Pt1 && Pt4 yra teisinga\n";
    else cout << "Pt1 && Pt4 yra neteisinga\n";
    return 0;
 }
```

Šios programos vykdymo rezultatas:

Pt1 ≠ Pt2
Pt1 = Pt3
Pt1 && Pt3 yra teisinga
Pt1 && Pt4 yra neteisinga

Priskyrimo operatoriaus perkrovimas.

Priskyrimo operatoriaus perkrovimas lyginant su kitais binariniai operatoriais turi tam tikrų ypatumų:

- operatorinė priskyrimo funkcija negali būti skelbiama kaip globalinė (ne klasės narė);
- operatorinė priskyrimo funkcija nepaveldima išvestinėje klasėje;
- kairysis priskyrimo operandas keičia savo reikšmę, todėl operatorinė priskyrimo funkcija turi gražinti nuorodą į objektą, kuriam ji buvo iškviečiama. Lengviausiai tai padaryti tokiu gražinimo sakiniu: *return *this*;
- operatorinės priskyrimo funkcijos realizacijos kode turi būti „priskyrimo sau pačiam“ patikrinimas, t.y. reikia išvengti, pvz., tokių klaidingų situacijų:

  ob = ob;

  arba tokių
  *Coord ob, *pOb;*
  *ob = *pOb;*

  Tipinė tokio patikrinimo konstrukcija:

  *if (this == &rhs) return *this;*

  čia *rhs* reiškia dešinės priskyrimo pusės operandą (angl. *right hand side*).

  Taip pat gali būti panaudota ir tokia konstrukcija:

  *if (*this == rhs) return *this;*

  tačiau tokiu atveju atitinkama klasė turi turėti perkrautą santykio (lyginimo) operatorių, nes *if* sakinyje dabar lyginami objektai, o ne rodyklės.

Pavyzdys 3.18. Priskyrimo operatoriaus perkrovimas.

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

```
class Taskas
{
        int x, y;
   public:
        Taskas(int _x, int _y) { x=_x, y=_y; }
        void GautiKoord(int& _x, int& _y) {  _x=x, _y=y;}
        Taskas& operator=(Taskas& rhs);
        bool operator==(Taskas rhs);
};

Taskas&
Taskas :: operator= (Taskas&  rhs)
{
// Priskyrimo sau pačiam patikrinimas
      if  (this == &rhs) return *this;
       x = rhs.x;
       y = rhs.y;
       return *this;
};

bool
Taskas :: operator== (Taskas  rhs)
{
      return (x == ob.x && y == ob.y);
};
main ()
{
    Taskas  Pt1(10,20), Pt2(10,25), Pt3(10,20);
    Pt2= Pt1;

    if  (Pt1== Pt2) cout << "Pt1 = Pt2\n";
    else cout << "Pt1 ≠ Pt2\n";
    Pt3=Pt2=Pt1;

    if  (Pt1== Pt3) cout << "Pt1 = Pt3\n";
    else cout << "Pt1 ≠ Pt3\n";

    return 0;
 }
```

Šios programos vykdymo rezultatas:

```
Pt1 = Pt2
Pt1 = Pt3
```

Unarinių (vienviečių) operatorių perkrovimas.

Unariniai operatoriai naudoja tik vieną operandą. Kai unarinis operatorius perkraunamas klasėje, šis operandas yra objektas, todėl bendra operatorinės funkcijos forma yra tokia:

<tipas> operatorX();

Kai unarinio operatoriaus funkcija skelbiama kaip globalinė, jos forma yra tokia:

<tipas> operatorX(<parametro tipas> par);

Pavyzdys 3.19. Unarinio operatoriaus (ženklo keitimo) perkrovimas.

```
#include <iostream.h>
class Taskas
{
        int x, y;
   public:
        Taskas() { x=0, y=0; }
        Taskas(int _x, int _y) { x=_x, y=_y; }
        void GautiKoord(int& _x, int& _y) {  _x=x, _y=y;}
        Taskas operator+();
        Taskas operator−();
};

Taskas
Taskas :: operator+ ()
{
      x = +x;
      y = +y;
      return *this;
};

Taskas
Taskas :: operator−()
{
      x = −x;
      y = −y;
      return *this;
};
main ()
{
    int x,y;
    Taskas  PtA(−10,20);
    PtA= +PtA;                    //Perkrauto unarinio operatoriaus + iškvietimas
    PtA.GautiKoord(x,y);
```

```
    cout << "PtA.x =" << x <<  "  PtA.y =" << y << endl;
    PtA= -PtA;                    //Perkrauto unarinio operatoriaus - iškvietimas
    PtA.GautiKoord(x,y);
    cout << "PtA.x =" << x <<  "  PtA.y =" << y << endl;

    return 0;
 }
```

Šios programos vykdymo rezultatas:

PtA.x =−10 PtA.y =20

PtA.x =10   PtA.y =−20

Šie operatoriai gali būti perkrauti ir tokiomis globalinėmis (draugiškomis klasei) funkcijomis:

```
        friend Taskas operator+( Taskas& ob);
        friend Taskas operator−( Taskas& ob);
};
```

Taip paskelbtų funkcijų realizacijos pavyzdžiai:

```
Taskas operator+( Taskas& ob);
{
      ob.x = +ob.x;
      ob.y = +ob.y;
      return ob;
};
Taskas operator−( Taskas& ob);
{
      ob.x = −ob.x;
      ob.y = −ob.y;
      return ob;
};
```

Kaip jau minėta, kai unarinis operatorius perkraunamas klasėje, bendra operatorinės funkcijos forma yra tokia:

<tipas> operatorX();

Tokiu būdu, unarinis (vienvietis) operatorius neturi parametrų, nes jis naudoja tik vieną operandą, o šis operandas yra objektas. Tačiau, kai perkraunami inkremento ++ ir dekremento – – operatoriai reikia skirti priešdėlinę (++x) ir priesaginę (x++) šio operatoriaus formas (angl.

prefix ir *postfix*). Tam tikslui priesaginio operatoriaus atveju nurodomas fiktyvus *int* tipo parametras, pvz., modifikuojant Pavyzdį 3.19:

```
class Taskas
{      . . .
   public:
          . . .
          Taskas operator++();
          Taskas operator++(int);
};
          . . .
Taskas
Taskas :: operator++()
{
       x++;
       y++;
       return *this;
};
Taskas
Taskas :: operator++(int)
{
       Taskas  tempOb=*this;
       ++*this;
       return tempOb;
};
          . . .
```

Priesaginio operatoriaus atveju, operatorinė funkcija grąžina ne nuorodą, o patį objektą, kadangi negalima grąžinti nuorodą į laikiną objektą.

Apibendrinant operatorinių funkcijų pavidalą, kai operatorių perkrovimas skelbiamas klasėje, paminėtini tokie ypatumai:

- Operatorinės funkcijos grąžinama reikšmė gali būti šių trijų tipų:
    - Klasės objektas, kai operatorius nekeičia pirmojo objekto reikšmės. Šiuo atveju operatorinės funkcijos viduje skelbiamas laikinas klasės tipo objektas;
    - Nuoroda į klasės objektą, kai pirmojo objekto reikšmė keičiama. Šiuo atveju funkcijoje modifikuojamas esamos klasės objektas ir gražinama rodyklės *this* reikšmė: *return *this;*
    - Loginė reikšmė, kai perkraunami santykio (lyginimo) operatoriai.

- Kai perkraunamas binarinis (dvivietis) operatorius, visada yra vienas parametras (pvz. *Taskas& ob*).
- Kai perkraunamas unarinis (vienvietis) operatorius, parametrų nėra, išskyrus priesaginę (angl. *postfix*) inkremento/dekremento formą.

Tipų transformavimo funkcijos.

Kaip jau minėta programuotojui suteikiama galimybė naudoti tipų suvedimo (transformavimo) konstruktorius. Tačiau, tokie konstruktoriai turi apribojimą: jie gali transformuoti standartinį tipą į objektą, bet ne atvirkščiai. Pastarąją galimybę suteikia tipų transformavimo funkcijos (operatoriai).

Bendra tokių funkcijų sintaksė :

operator<naujo tipo vardas> ();

Tad, tipų transformavimo funkcijos neturi parametrų ir grąžinamos reikšmės tipą (manoma, kad jos grąžina tipą, nurodytą po žodžio **operator**).

Pavyzdys 3.20. Tipų transformavimo funkcijos (operatoriaus) taikymas.

```
#include <iostream.h>
class Taskas
{
        int x, y;
   public:
        Taskas() { x=0, y=0; }
        Taskas(int _x, int _y) { x=_x, y=_y; }
        //Tipo transformavimo funkcija
        operator int() {return x*y;}
};

main ()
{
    int n;
    Taskas  Pt(10,20);
    n= Pt + 30;       //Neišreikštinis tipo transformavimo funkcijos iškvietimas
    //Tiesioginis tipo transformavimo funkcijos iškvietimas
    cout << "int(Pt) = " << int(Pt) << endl;
```

```
    cout << "n = Pt + 30   = " << n << endl;
}
```

Šios programos vykdymo rezultatas:

```
int(Pt) = 200
n = Pt + 30   = 230
```

Tokiu būdu, tipo transformavimo funkcijos naudojimas leidžia objektui Pt dalyvauti tiek neišreikštinio, tiek tiesioginio tipo transformavimo veiksmuose.

ŠABLONAI IR PARAMETRIZAVIMAS

Kaip jau minėta, šablonai leidžia aprašyti apibendrintas funkcijas ir klases, kurios galima panaudoti su įvairiais duomenimis. Konkrečių duomenų tipų pateikimas šabloninei funkcijai ar klasei vadinamas parametrizavimu, o pačios funkcijos (klasės) vadinamos parametrizuotomis.

Funkcijos šablono skelbimo sintaksė:

template <class T_1, class T_2, …, class T_n>

<tipas> <funkcijos_vardas> (parametrų sąrašas)

{

// funkcijos tekstas

}

Kiekvienas šablono parametras aprašomas arba baziniu žodžiu *class* ir tipo vardu, arba tipo vardu ir identifikatoriumi. Parametrizuotiems tipams aprašyti vietoj *class* galima naudoti bazinį žodį *typename*.

Šablonai leidžia sukurti apibendrintas funkcijas ir klases, neprisirišant prie konkrečių duomenų tipų. Jie taip pat naudojami kuriant universalias bibliotekas (pvz., STL) ir tokiu būdu užtikrina daugkartinį funkcijų ar klasių panaudojimą.

Pavyzdys 3.21. Šabloninių funkcijų naudojimas C++.

```
#include <iostream>
using namespace std;
```

```
template <class T>
T Sqr(T x)
{
      return x*x
}

template <class T>
T* Swap(T* t, int ind1, int ind2)
{
      T temp = t[ind1];
      t[ind1] = t[ind2];
      t[ind2] = temp
      return t;
}

int main ()
{
    int n=10, sq_n, i = 2, j = 5;
    double d = 20.10, sq_d;
    char* str = „Šablonas“;

    sq_n = Sqr(n);
    sq_d = Sqr(d);

    cout << “reiksme n =” << n <<  endl << “n kvadratas =”<<sq_n << endl;
    cout << “reiksme d =” << d <<  endl << “d kvadratas =”<<sq_d << endl;
    cout << “pradine eilute = ‘ ” << str <<  “ ‘ “ << endl;
    cout << “pakeista eilute = ‘ ” << Swap(str, i, j) <<  “ ‘ “ << endl;
    return 0;
 }
```

Šios programos vykdymo rezultatas:

reiksme n = 10

n kvadratas = 100

reiksme d = 20.10

d kvadratas = 404.01

pradine eilute = ‘Šablonas‘

pakeista eilute = ‘Šanlobas‘

Kiekvienas šabloninės funkcijos iškvietimas – tai nurodymas kompiliatoriui sugeneruoti funkcijos kodą konkrečiam duomenų tipui. Todėl šis procesas, pvz., *sq_d = Sqr(d)*, vadinamas šabloninės funkcijos *konkretizavimu*.