C++ programavimo kalba

Konstruktorius, destruktorius,
klasių metodų modifikatoriai, objektų masyvai
(4 paskaita)

Konstruktorius

Sukuriant objektą, jo duomenims paprastai turi būti priskiriamos pradinės reikšmės. Tai gali atlikti tam skirti metodai. Bet būtų patogu, jei kuriant objektą, duomenų inicializacija būtų atliekama **automatiškai**, neiškviečiant papildomų metodų.

Tai galima atlikti naudojant specialų metodą – *konstruktorių*.

Konstruktorius – tai metodas, kuris automatiškai vykdomas kiekvieną kartą, kai tik sukuriamas objektas.

Konstruktoriaus savybės:

- Turi tokį pat pavadinimą, kaip ir klasė;
- Nieko negrąžina t.y. neturi operatoriaus *return*;
- Konstruktoriaus paskirtis – priskirti pradines reikšmes klasės kintamiesiems.

Konstruktorius

Klasė 

```
#include <iostream>
using namespace std;
class studentas
{ private:   float svoris;
  public:
     studentas(float x)          // konstruktorius
     { svoris = x;  }
     void rodo()
     { cout << svoris << endl; }
};
void main()
{ studentas Algis(120.25);
  studentas Jonas(56.2);
  cout<<“Algio svoris “; Algis.rodo();
  cout<<“Jono svoris “; Jonas.rodo();}
```

Konstruktorius

```
#include <iostream>
using namespace std;
class Counter
{private:   unsigned int count;
 public:
     Counter()
     { count = 0;  }
     void increase()
     { count ++ ; }
     int show()
     { return count; }
};
```

```
void main()
{ Counter C1;
  cout<<“Skaitliukas = “<<
              C1.show() << endl;
C1.increase();
cout <<“Skaitliukas =“ <<
             C1.show()<<endl;
}
```

Konstruktorius

Konstruktoriuje klasės duomenims reikšmės gali būti priskiriamos naudojant įprastinę funkcijų sintaksę arba inicializacinį sąrašą.

Naudojant inicializacinį sąrašą, kintamiesiems reikšmės priskiriamos prieš konstruktoriaus vykdymą. *const* tipo duomenys bei nuorodos gali būti inicializuotos tik inicializaciniame sąraše.

```
Counter()
{ count = 0; }
```

Konstruktorius kaip funkcija

```
Counter() : count(0)
{ }
```

Automatinė inicializacija per sąrašą

Inicializacinis sąrašas

```
class circle
{ private:
            int radius;
            int c_x, c_y;
  public:
            circle (int a, int b, int c) : radius(a), c_x(b), c_y(c)
            { }
};
```

Apibendrinus

```
AnyClass( ) : m1(7), m2(12), m3(23.56)
   { }

AnyClass( int x, int y, float z ) : m1(x), m2(y), m3(z)
   { }
```

Konstruktoriaus savybės

- Konstruktorius negrąžina jokios reikšmės;
- Nenurodomas konstruktoriaus kaip funkcijos tipas;
- Konstruktorius nepaveldimas;
- Konstruktorius negali būti apibrėžtas taikant modifikatorius *const, violatile, static, virtual*;
- Jei klasėje neapibrėžtas konstruktorius, kuriant objektą, vykdomas konstruktorius pagal nutylėjimą.

Konstruktorių perkrovimas (overloading)

Jei klasėje nėra apibrėžtas konstruktorius, kompiliatorius automatiškai kuria konstruktorių be argumentų sąrašo. Toks konstruktorius rezervuoja atmintį objektui, bet duomenims reikšmių nepriskiria.

Visgi tikslinga atskirai sukurti konstruktorių be argumentų sąrašo, priskiriant duomenims pradines reikšmes.

Jei klasėje yra keli konstruktoriai, sakoma, kad konstruktorius yra perkrautas (overloaded).

```
class atstumas
{ private:  float ilgis; float plotis;
  public:  atstumas() : ilgis(0.0), plotis(0.0)
  { }
  atstumas (float x, float y)
  { ilgis = x;   plotis = y; }
};
```

```
void main()
{  atstumas a1, a2;
    atstumas a3(12.4, 45.1);
}
```

Pavyzdys

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

class Staciakampis {
    int ilgis, plotis;
  public:
    Staciakampis (): ilgis(3), plotis(3)
    { }
    Staciakampis (int, int);
    int plotas () { return (ilgis*plotis); }
};

Staciakampis :: Staciakampis (int a, int b)
{
  ilgis = a;
  plotis = b;
}
```

```
int main () {
 Staciakampis rect;
 Staciakampis rectb (5,6);
  cout << “Plotas: " << rect.plotas()
          << endl;
  cout << “Plotas: " <<
                  rectb.plotas() << endl;
  return 0;
}
```

Konstruktorius pagal nutylėjimą

Jei klasėje nėapibrėžtas nei vienas konstruktorius, tuomet kompiliatorius sukuria konstruktorių pagal nutylėjimą. Toks konstuktorius nepriskiria objekto duomenims jokios reikšmės, nors atmintyje išskiria kintamiesiems vietą. Konstruktorius pagal nutylėjimą naudojamas tik tuomet, kai nėra sukuriamas joks kitas konstruktorius.

```
class Staciakampis
{
    int ilgis, plotis;
public:
    int plotas (int a, int b)
 {
    ilgis = a;   plotis = b;
    return (ilgis*plotis); }
};
```

```
class Staciakampis
 {   int ilgis, plotis;
  public:
    Staciakampis (): ilgis(2), plotis(3)
    { }
    int plotas () {
    return (ilgis*plotis); }
};
Staciakampis didelis;
```

Kopijos konstruktorius

(default copy constructor)

Objektas gali būti inicializuotas naudojant kito to paties tipo objekto kopiją. Tokiu atveju nereikia kurti specialaus konstruktoriaus, jį automatiškai kuria kompiliatorius.

Kopijos konstruktorius – tai vieno argumento funkcija, kurios argumentas tos pačios klasės objektas.

```
class distance
{ float ilgis; float plotis;
   public:
        distance (float x, float y)
        { ilgis = x;
          plotis = y; }
    void show()
{ cout<< "Kintamieji ilgis " << ilgis<<endl;
  cout<< “\t plotis " << plotis<<endl;
 } };
```

```
void main ()
{  distance d3(2.5, 5.8);
   distance d2(d3);       // kopijos konstr.
   distance d1 = d3;      // kopijos konstr.
   d3.show(); d2.show(); d1.show();
  }
```

Destruktorius

Destruktorius – tai specialus metodas, kuris iškviečiamas automatiškai tuomet, kai naikinamas objektas. Jo paskirtis naikinti objektą ir atlaisvinti atmintį.

Destruktoriaus savybės:

- pavadinimas “~klasės_pavadinimas”
- nieko negrąžina
- neturi argumentų sąrašo
- destruktorius nepaveldimas
- destruktorius negali būti apibrėžtas naudojant modifikatorius *const, violatile, static, virtual*

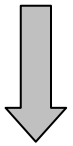
Jei destruktorių reikia iškviesti rankomis, naudojama sintaksė:

objektas.~klasė();

Destruktorius (pavyzdys)

```
#include <iostream>
using namespace std:
class Any
{ private:  int a, b;
  public:
     Any(int x, int y)
      { a = x;
        b = y;  }
      void show_x()
      { cout << a << endl;
        cout<<  b << endl;}
     ~Any()
      {cout<<“Vykdomas destruktorius \n”;
      }
};
```

```
void main()
{ Any c1(33, 22);
  c1.show_x();
 }
```



Ekrane

```
33
22
Vykdomas destruktorius
```

Destruktorius (pavyzdys)

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

class Staciakampis {
    int *ilgis, *plotis;
  public:
Staciakampis (int, int);
~Staciakampis ();
    int plotas () { return (ilgis*plotis); }
};

Staciakampis :: Staciakampis (int a, int b)
{ ilgis = new int; plotis = new int;
  *ilgis = a;
  *plotis = b;
}
```

```
Staciakampis ::~ Staciakampis () {
delete width; delete height; }


int main () {
 Staciakampis rect (3,3);
 Staciakampis rectb (5,6);
  cout << “Plotas: " << rect.plotas()
          << endl;
  cout << “Plotas: " <<
                  rectb.plotas() << endl;
  return 0;
}
```

Rodyklė **this**

Kiekvienas C++ objektas turi specialią rodyklę, kurios pavadinimas **this**. Ją automatiškai sukuria kompiliatorius. Rodyklė **this** rodo į patį objektą.

Rodyklės this veikimo sritis – klasė, kurioje ji apibrėžta.

Faktiškai this yra paslėptas klasės parametras, kurį prideda pats kompiliatorius metodo iškvietimo metu. Metodo iškvietimo pavyzdys:

```
Obj.name(par1, par2);       ->    Obj.name(&Obj, par1, par2);

Obj::name(param1, param2)
 { cout<<“Hello”<<param1<<endl;
  cout<<“Hello”<<this->param1<<endl;     // tas pats,kaip ankst.eilutėje
  cout << this<< endl; }                 // išvedamas objekto adresas
```

Dažniausiai this naudojamas, kai metodas gražina rodyklę arba nuorodą į objektą pvz: return this; return *this;

Tai pritaikoma operatorių perkrovime.

Statiniai klasės duomenys

Statiniai duomenys klasėse naudojami, kai visi tos pačios klasės objektai turi dalintis bendra informacija. Pavyzdys – programoje sukurtų objektų skaičius.

Statinių duomenų apibrėžimui naudojamas raktinis žodis static. Toks kintamasis pasiekiamas tik iš klasės, tačiau jis atmintyje saugomas iki programos darbo pabaigos.

```
class pva
{ private: static int count;
  public: pva ()
          { count ++;}
          int getcount ()
          { return count; }
};
```

```
int pva::count = 0;   // tik kai static
void main()
{ pva f1, f2, f3;
  cout<< f1.getcount()<<endl;  // 3
  cout<< f2.getcount()<<endl;  // 3
  cout<< f3.getcount()<<endl;  // 3
 }
```

const ir klasės

Klasės duomenys negali būti konstantos!!!

Tik klasės metodai gali būti apibrėžti su modifikatoriumi const

Toks metodas negali modifikuoti klasės kintamųjų reikšmių, todėl const metodai **naudojami tik reikšmių išvedimui**. Tokių metodų argumentų sąraše dažniausiai būna nuorodos.

```
class aClass
   {  private: int alpha;
      public:
          aClass() : alpha(10)
          { }
          void nonFunc()
          { alpha = 100; }
         void consFunc() const
          { alpha = 200;            // klaida, negalima modifikuoti
            cout <<alpha<<endl; }
   } ;
```

const ir objektai

Galima apibrėžti ir **const objektus**. Tuo atveju objekto negalima modifikuoti, tai reiškia kad galima naudoti tik metodus, apibrėžtus, kaip **const**, nes tik jie garantuoja, kad duomenys nebus modifikuoti.

```
class Distance
    {  private: float ilgis; float plotis;
       public:
            distance (float x, float y) : ilgis(x), plotis(y)
            {  }
     void show() const
     { cout<< "Kintamieji ilgis " << ilgis<<endl;
       cout<< “\t plotis " << plotis<<endl;
     }
    void get()
    {cin>>ilgis>>plotis;}
};
```

```
void main()
{const Distance d(2.3, 4.3);
d.get();       // klaida
d.show();
}
```

Objektų masyvai

Objektai kaip ir paprasti kintamieji gali sudaryti masyvus. Sintaksė niekuo nesiskiria nuo įpratinės masyvų apibrėžimo sintaksės.

AnyClass ACDC[10];

Kompiliatorius kurdamas masyvą, naudoja konstruktorių pagal nutylėjimą, todėl rekomenduojama klasėje apibrėžti tokį konstrukorių. Masyvų elementai pasiekiami analogiškai, kaip ir įprastinių masyvų.

```
class AnyClass
{ int a;
    public:
    AnyClass(int x) { a = x; }
    int show()
        { retunt a; }
};
```

```
main()
{ AnyClass Arr[3] = { 12, 14, 16};
for (int i = 0; i<3; i++)
        cout<< Arr[i].show()<< endl;
return 0; }
```

Objektų masyvai (pavyzdys)

```
#include <iostream>
using namespace std;
class plotas
 { private: int x, y, stac_plotas;
   public:  void calc(int a, int b)
                { x = a; y = b; stac_plotas = a*b;}
                int getcalc ()
                { return stac_plotas; }
};

void main()
{ plotas f1[5];                    // sukuriamas 5 objektų masyvas
  for (int i = 0; i < 5; i++)
    {  f1[i].calc(i+1, i+3);        // priskiriamos reikšmės ir skaičiuojami plotai
       cout << f1[ i ].getcalc() <<endl;
    }
}
```

Objektų masyvai (pavyzdys)

```
class AnyClass
{       int x, y;
    public:
        AnyClass(int a, int b) { x=a, y=b; }
        int Getx() { return x; }
        int Gety() { return y; }
};

main ()
{ AnyClass Arr[2][2] = { AnyClass(1,2), AnyClass(2,3), AnyClass(3,4),
                         AnyClass(5,6)};
int i , j;
for (i = 0; i<2; i++)
for (j = 0; j<2; j++)
     { cout<< Arr[i][j].Getx();
       cout<< Arr[i] [j].Gety()<< endl;}
return 0; }
```

Kas bus ekrane?

Objektų masyvai (pavyzdys)

```
class AnyClass
{ int x, y;
   public:
AnyClass(int a, int b) { x=a, y=b; }
AnyClass() { x=0, y=0; }                    // konstruktorius pagal nutylėjimą
int Getx() { return x; }
int Gety() { return y; }
};
main ()
{ AnyClass * ptr;
int i ;
ptr = new AnyClass[6];
for (i = 0; i < 6; i++)
     { cout<< ptr->Getx()<< endl;
       cout<< ptr->Gety()<< endl;
        ptr ++;  }
return 0; }
```

Kas bus ekrane?