Objektinis Programavimas

Operatorių persidengimo tęsinys & copy ir move semantika

Iš čia kylama į žvaigždes

© dr. Remigijus Paulavičius, remigijus.paulavicius@mii.vu.lt

Turinys

Vector klasė (1)

operator[] realizacija

```
#include <iostream>
class Vector {
 private:
  int sz;
  double* elem;
 public:
  Vector(int s, double val) : sz(s), elem(new double[sz]) { std::fill_n(elem, s, val); }
  int size() const { return sz; }
  double& operator[](int i) { return elem[i]; } // Ar čia viskas gerai?
};

int main() {
  Vector v1 {2, 1.0}; // sukonstruojame vektorių: (1.0, 1.0)
  std::cout << v1[1] << ", " << v1[2] << std::endl; // Ką gausime?
}
```

Vector klasė (2)

operator[] realizacija

```
#include <iostream>
class Vector {
 private:
  int sz;
  double* elem;
 public:
  Vector(int s, double val) : sz(s), elem(new double[sz]) { std::fill_n(elem, s, val); }
  int size() const { return sz; }
  double& operator[](int i) {
    if (i < 0 || size() <= i) throw std::out_of_range {"Vector::operator[]"};
    return elem[i];
  }
};

int main() {
  Vector v1 {2, 1.0}; // sukonstruojame vektorių: (1.0, 1.0)
  std::cout << v1[1] << ", " << v1[2] << std::endl; // Ką gausime?
}
```

Vector klasė (3)

operator[] realizacija

```
#include <iostream>
class Vector {
 private:
  int sz;
  double* elem;
 public:
  Vector(int s, double val) : sz(s), elem(new double[sz]) { std::fill_n(elem, s, val); }
  int size() const { return sz; }
  double& operator[](int i) {
    if (i < 0 || size() <= i) throw std::out_of_range {"Vector::operator[]"};
    return elem[i];
  }
};

int main() {
  Vector v1 {2, 1.0}; // sukonstruojame vektorių: (1.0, 1.0)
  std::cout << v1[0] << ", " << v1[1] << std::endl; //
  const Vector v2 {2, 2.0}; // sukonstruojame const vektorių: (2.0, 2.0)
  std::cout << v2[0] << ", " << v2[1] << std::endl; // Ką gausime?
}
```

Vector klasė (4)

operator[] realizacija

```cpp
#include <iostream>
class Vector {
 private:
  int sz;
  double* elem;
 public:
  Vector(int s, double val) : sz(s), elem(new double[sz]) { std::fill_n(elem, s, val); }
  int size() const { return sz; }
  double& operator[](int i) {
    if (i < 0 || size() <= i) throw std::out_of_range {"Vector::operator[]"};
    return elem[i];
  }
  const double& operator[](int i) const { return elem[i]; } // Pilna versija su throw
};

int main() {
  Vector v1 {2, 1.0}; // sukonstruojame vektorių: (1.0, 1.0)
  std::cout << v1[0] << ", " << v1[1] << std::endl; //
  const Vector v2 {2, 2.0}; // sukonstruojame const vektorių: (2.0, 2.0)
  std::cout << v2[0] << ", " << v2[1] << std::endl; // Ką gausime?
}
```

Objektų kopijavimas (1)

— Objektai, kaip ir fundamentalieji kintamieji, gali būti nukopijuoti.

— Numatytasis (default) kopijavimas vyksta perkopijuojant kiekvieną objekto narį.

— **Visuomet turime įvertinti**, ar objektas gali ir jei gali, tai kaip gali būti kopijuojamas?

```
#include <iostream>
#include "Complex.h"
int main() {
  Complex c1 {1.0, 1.0};  // konstruktorius: Complex(double, double)
  Complex c2 {c1};        // copy-initialization
  Complex c3;             // default konstruktorius
  c3 = c1;                // copy-assignment
  std::cout << "c1: " << c1 << "c2: " << c2 << "c3: " << c3 << std::endl; // Ką gausime?
}
```

Objektų kopijavimas (2)

— Paprastiems (konkretiems) tipams panariui (memberwise) kopijavimas dažniausiai yra teisinga kopijavimo semantika (strategija).

— Tačiau sudėtingesniems tipams panariui kopijavimas dažnai yra klaidinga strategija.

— Kai klasė yra atsakinga už resursų valdymą (t.y. kai klasė atsakinga už objektą pasiekiamą per rodyklę), panariui kopijavimas dažniausiai yra didelės nelaimės pradžia. **Kodėl?**

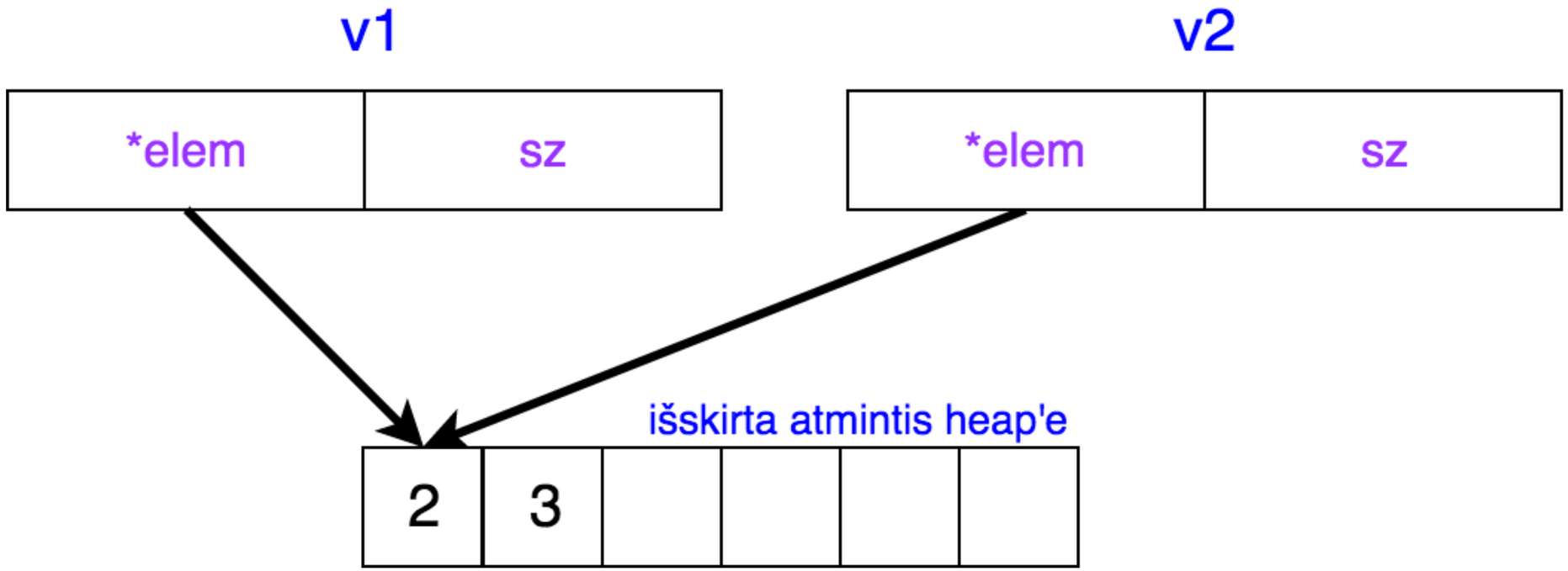
Objektų kopijavimas (3)

Vector klasės objektų kopijavimas

```
#include <iostream>
class Vector {
 private:
  int sz;
  double* elem;
 public:
  Vector(int s, double val) : sz(s), elem(new double[sz]) { std::fill_n(elem, s, val); }
  double& operator[](int i) { return elem[i]; }
  const double& operator[](int i) const { return elem[i]; }
};
int main() {
  Vector v1 {2, 1.0};  // sukonstruojame vektorių: (1.0, 1.0)
  Vector v2 = v1;      // nukopijuojame v1 vektorių
  v1[0] = 2.0;         // turėtų būti (2.0, 1.0)?
  v2[1] = 3.0;         // turėtų būti (1.0, 3.0)?
  std::cout << v1[0] << ", " << v1[1] << std::endl; // Ką gausime?
  std::cout << v2[0] << ", " << v2[1] << std::endl; // Ką gausime?
}
```

Objektų kopijavimas (4)

Vector'ių kopijavimo (memberwise**) iliustracija**



— Bet jeigu taip daryti blogai, kodėl (po 😈) kompiliatorius neįspėjo?

Objektų kopijavimas (5)

— Ar prisimenate C++ Garbage collector ?

The technique of acquiring resources in a constructor and releasing them in a destructor, known as Resource Acquisition Is Initialization or RAII.
— Bjarne Stroustrup

— Jei Vector turėtų **destruktorių**, kompiliatorius turėtų bent įspėti, kad memberwise semantika yra neteisinga!

Objektų kopijavimas (6)

```
#include <iostream>
class Vector {
 private:
  int sz;
  double* elem;
 public:
  Vector(int s, double val) : sz(s), elem(new double[sz]) { std::fill_n(elem, s, val); }
  ~Vector() { delete[] elem; }
  double& operator[](int i) { return elem[i]; }
};
int main() {
  Vector v1 {2, 1.0};  // sukonstruojame vektorių: (1.0, 1.0)
  Vector v2 = v1;      // nukopijuojame v1 vektorių
  v1[0] = 2.0;
  v2[1] = 3.0;
  std::cout << v1[0] << ", " << v1[1] << std::endl; // Ką dabar gausime?
  std::cout << v2[0] << ", " << v2[1] << std::endl; // Ką dabar gausime?
}
```

Objektų kopijavimas (7)

— Klasės objekto kopijavimą apibrėžia **kopijavimo konstruktorius** (copy constructor) ir kopijavimo **priskirties operatorius** (copy assignment) 🙀

```
class Vector {
private:
  double* elem;
  int sz;
public:
  Vector() : sz(0), elem(new double[sz]) {}  // konstruktorius išskiria resursus
  Vector(int s) : sz{ s }, elem{ new double[sz] } { std::fill_n(elem, s, 0.0); }
  Vector(int s, double val)
  ~Vector() { delete[] elem; }               // destruktorius: atlaisvina resursus

  Vector(const Vector& v);                   // copy konstruktorius
  Vector& operator=(const Vector& v);        // priskyrimo operatorius

  double& operator[](int i);
  int size() const;
};
```

Copy konstruktorius (1)

```
// Vector.cpp realizacija
Vector::Vector(const Vector& v) // copy konstruktorius
    : sz{v.sz},                 // inicializuojame sz
      elem{new double[v.sz]}    // išskiriame atmintį elem
{
  for (int i=0; i!=sz; ++i)   // nukopijuojame elementus
    elem[i] = v.elem[i];
}
```

— Ar pastebite kažką neįprasto?

Copy konstruktorius (2)

```
// Vector.cpp realizacija
Vector::Vector(const Vector& v) // copy konstruktorius
    : sz{v.sz},                 // inicializuojame sz
      elem{new double[v.sz]}    // išskiriame atmintį elem
{
  for (int i=0; i!=sz; ++i)   // nukopijuojame elementus
    elem[i] = v.elem[i];
}
```

You can access private members of a class from within the class, even those of another instance. [access]

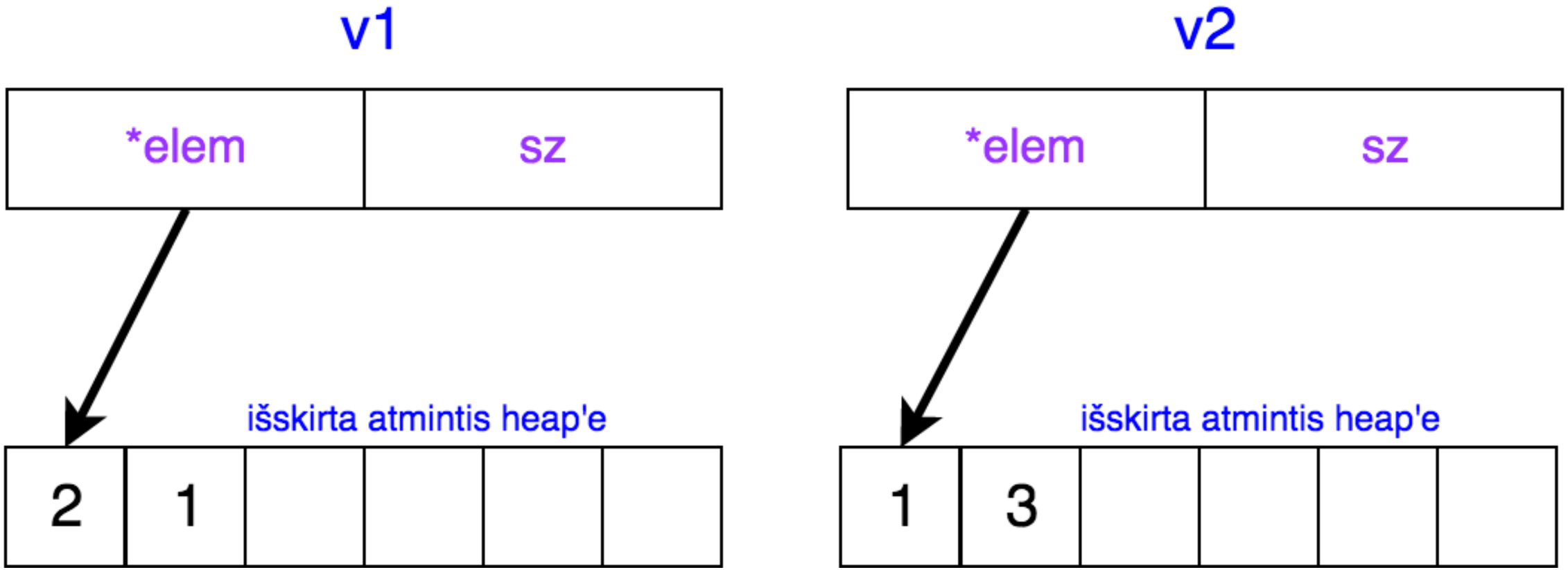
[access] https://stackoverflow.com/a/4117020/3737891

Objektų kopijavimas (8)

```
#include <iostream>
class Vector {
 private:
  int sz;
  double* elem;
 public:
  Vector() : sz(0), elem(new double[sz]) {}  // default konstruktorius
  Vector(int s, double val) : sz(s), elem(new double[sz]) { std::fill_n(elem, s, val); }
  ~Vector() { delete[] elem; }
  Vector(const Vector& v) :elem{new double[v.sz]}, sz{v.sz} { // copy konstruktorius
    for (int i=0; i!=sz; ++i) elem[i] = v.elem[i];
  }
  double& operator[](int i) { return elem[i]; }
};
int main() {
  Vector v1 {2, 1.0};  // sukonstruojame vektorių: (1.0, 1.0)
  Vector v2 = v1;      // copy konstruktorius kopijuoja
  v1[0] = 2.0;
  v2[1] = 3.0;
  std::cout << v1[0] << ", " << v1[1] << std::endl; // Ką dabar gausime?
  std::cout << v2[0] << ", " << v2[1] << std::endl; // Ką dabar gausime?
}
```

Objektų kopijavimas (9)

Vector'ių kopijavimas: copy konstruktorius



Objektų kopijavimas (10)

```
int main() {
  Vector v1 {2, 1.0};  // sukonstruojame vektorių: (1.0, 1.0)
  Vector v2 {v1};      // (ar tik) kita sintaksė?
  v1[0] = 2.0;
  v2[1] = 3.0;
  std::cout << v1[0] << ", " << v1[1] << std::endl; // Ką dabar gausime?
  std::cout << v2[0] << ", " << v2[1] << std::endl; // Ką dabar gausime?
}
```

— O ką dabar gausime?

Objektų kopijavimas (11)

```
int main() {
  Vector v1 {2, 1.0};  // sukonstruojame vektorių: (1.0, 1.0)
  Vector v2;           // default konstruktorius
  v2 = v1;             // v2 priskiriame vektorių v1
  v1[0] = 2.0;
  v2[1] = 3.0;
  std::cout << v1[0] << ", " << v1[1] << std::endl; // Ką dabar gausime?
  std::cout << v2[0] << ", " << v2[1] << std::endl; // Ką dabar gausime?
}
```

— O ką dabar gausime?

Priskyrimo operatorius (operator=)

```
// Vector.cpp realizacija
Vector& Vector::operator=(const Vector& v) { // priskyrimo operatorius
    // Savęs priskyrimo aptikimas
    if (&v == this) return *this;

    double* p = new double[v.sz];
    for (int i=0; i!=v.sz; ++i) // nukopijuojame v elementus
        p[i] = v.elem[i];
    delete[] elem; // atlaisviname seną atmintį!
    elem = p;      // elem point'ina į naują atmintį
    sz = v.sz;     // atnaujiname size
    return *this;  // grąžiname objektą
}
```

Objektų kopijavimas (12)

Naudojant priskyrimo operatorių

```
// Includinti Vector.h su realizuotu operator=
int main() {
  Vector v1 {2, 1.0};  // sukonstruojame vektorių: (1.0, 1.0)
  Vector v2;           // default konstruktorius
  v2 = v1;             // nukopijuojame v1 per operator=
  v1[0] = 2.0;
  v2[1] = 3.0;
  std::cout << v1[0] << ", " << v1[1] << std::endl; // Ką dabar gausime?
  std::cout << v2[0] << ", " << v2[1] << std::endl; // Ką dabar gausime?
}
```

Rule of three[r3]

The rule of three (the Law of The Big Three) is a rule of thumb in C++ (prior to C++11) that claims that if a class defines one (or more) of the following it should probably explicitly define all three:

— destructor

— copy constructor

— copy assignment operator

[r3] https://en.wikipedia.org/wiki/Ruleofthree(C%2B%2Bprogramming)