Programowanie Obiektowe

C++ operatory i przyjaciele

Dariusz Brzeziński

Politechnika Poznańska, Instytut Informatyki

Funkcje operatorowe

- Pozwalają na przeciążenie istniejących operatorów, tak aby uzyskały znaczenie dla definiowanych przez programistę nowych typów obiektowych
- Można przeciążyć m.in.:
  - +, -, *, /, !, =, <, >, ++, --, (), [], new, delete
- Nie można zmienić priorytetów operatorów, np. a+b*c ≡ a+(b*c)
- Nie można zmienić wymaganej liczby argumentów operatorów, np.
  - / zawsze 2-argumentowe,
  - ! zawsze 1-argumentowa
- Mogą być definiowane jako metody klasy lub funkcje zewnętrzne (najczęściej zaprzyjaźnione)
- Nazwa funkcji operatorowej jest konkatenacją słowa kluczowego operator i symbolu operatora np. operator=, operator++, ...
- Przynajmniej jeden argument wywołania operatora musi być typu zdefiniowanego przez użytkownika, np. operator+(int, float) jest niemożliwe
- Operatory są dziedziczone

Funkcje operatorowe (2)

```
class Complex
{
    int re, im;

public:
    Complex() {};
    Complex(int x, int y) : re(x), im(y) {}
    Complex& operator++()  // przedrostkowy
    { re++; return *this; }
    Complex& operator++(int i)  // przyrostkowy
    { im++; return *this; }
    Complex& operator=(const Complex &c)
    {
        re = c.re;
        im = c.im;
        return *this;
    }

    operator int() { return re; }

    // operator wywołania funkcji
    void operator()(int x) { re += x; }
    void operator()(int x, int y)
    { re += x; im += y; }

    friend Complex operator+(Complex c1, Complex c2);
    friend ostream& operator<<(ostream &o, Complex c);
};
```

```
Complex operator+(Complex c1, Complex c2)
{
    return Complex(c1.re + c2.re, c1.im + c2.im);
}

ostream& operator<<(ostream &o, Complex c)
{
    o << "[" << c.re << ";" << c.im << "]";
    return o;
}

void main()
{
    Complex z1 = Complex (1, 2);
    Complex z2 = Complex (2, 3);
    Complex z3;
    z3 = z1 + z2;
    cout << z3;     // [3;5]
}
```

Funkcje operatorowe (3)

```
class String
{
    char *txt;
public:
    // konstruktory itp. ...

    char& operator[](int idx)
    {
        return *(txt + idx);
    }
};
```

Operator indeksowania z referencją zwraca l-wartość.
Bez referencji zwracałby znak bez możliwości jego zmiany.

Funkcje operatorowe (4)

```
class wektor
{
public:
    float x, y, z;
    wektor operator*(float i)
    {
        return wektor(this->x*i, this->y*i, this->z*i);
    };
    wektor(float p_x, float p_y, float p_z): x(p_x), y(p_y), z(p_z){};
    wektor()
    {
        x = 0;
        y = 0;
        z = 0;
    };
};

void main()
{
    wektor w1 = wektor(1, 1, 1);
    w1 = w1*3;
    // w1=3*w1;  wywołanie niepoprawne
}
```

Funkcja operatorowa zdefiniowana jako metoda klasy wymaga, aby obiekt stojący po lewej stronie operatora był obiektem klasy.
Operator, który jest funkcją globalną nie ma tego ograniczenia.

Funkcje operatorowe (5)

```
class wektor
{
public:
    float x, y, z;
    friend wektor operator*(wektor w, float i);
    friend wektor operator*(float i, wektor w);
    wektor(float p_x, float p_y, float p_z): x(p_x), y(p_y), z(p_z){};
    wektor()
    {
        x = 0;
        y = 0;
        z = 0;
    };
};

wektor operator*(wektor w, float i)
{
    return wektor(w.x*i, w.y*i, w.z*i);
};

wektor operator*(float i, wektor w)
{
    return wektor(w.x*i, w.y*i, w.z*i);
};

void main()
{
    wektor w1 = wektor(1, 1, 1);
    w1 = w1*3;
    w1 = 3*w1;
}
```

Funkcje operatorowe (6)

```
class C1
{
    int a;
    char tekst[20];

public:
    C1(int p_a, char *p_tekst): a(p_a)
    {
        strcpy(tekst, p_tekst);
    };
    C1()
    {
        a = 0;
        strcpy(tekst, "pusty obiekt");
    };
    void wartosc()
    {
        cout << a << " " << tekst << endl;
    }
    void set_tekst (char *p_tekst)
    {
        strcpy(tekst, p_tekst);
    };
};
```

```
void main()
{
    C1 obj1(1, "to jest obiekt 1");
    C1 obj2;
    cout << "obj1 = "; obj1.wartosc();
    cout << "obj2 = "; obj2.wartosc();
    obj2 = obj1;
    cout << "obj2 = "; obj2.wartosc();
    obj1.set_tekst("zmiana tekstu");
    cout << "obj2 = "; obj2.wartosc();
}
```

Wynik:

```
obj1 = 1 to jest obiekt 1
obj2 = 0 pusty obiekt
obj2 = 1 to jest obiekt 1
obj2 = 1 to jest obiekt 1
```

Funkcje operatorowe (7)

```
class C1
{
    int a;
    char *tekst;

public:
    C1(int p_a, char *p_tekst): a(p_a)
    {
        tekst = new char[strlen(p_tekst)+1];
        strcpy(tekst, p_tekst);
    };
    C1()
    {
        a = 0;
        tekst = new char[13];
        strcpy(tekst, "pusty obiekt");
    };
    void wartosc()
    {
        cout << a << " " << tekst << endl;
    }
    void set_tekst (char *p_tekst)
    {
        strcpy(tekst, p_tekst);
    };
};
```

```
void main()
{
    C1 obj1(1, "to jest obiekt 1");
    C1 obj2;
    cout << "obj1 = "; obj1.wartosc();
    cout << "obj2 = "; obj2.wartosc();
    obj2 = obj1;
    cout << "obj2 = "; obj2.wartosc();
    obj1.set_tekst("zmiana tekstu");
    cout << "obj2 = "; obj2.wartosc();
}
```

Wynik:

```
obj1 = 1 to jest obiekt 1
obj2 = 0 pusty obiekt
obj2 = 1 to jest obiekt 1
obj2 = 1 zmiana tekstu
```

Funkcje operatorowe (8)

```
class C1
{
    int a;
    char *tekst;

public:
    C1(int p_a, char *p_tekst): a(p_a)
    {
        tekst = new char[strlen(p_tekst)+1];
        strcpy(tekst, p_tekst);
    };
    C1()
    {
        a = 0;
        tekst = new char[13];
        strcpy(tekst, "pusty obiekt");
    };
    ~C1()  { delete [] tekst; };
    void wartosc()
    {
        cout << a << " " << tekst << endl;
    }
    void set_tekst (char *p_tekst)
    {
        strcpy(tekst, p_tekst);
    };
    C1 & operator=(const C1 &zrodlo);
};
```

```
C1 & C1::operator=(const C1 &zrodlo)
{
    delete [] tekst;
    tekst = new char [strlen(zrodlo.tekst)+1];
    strcpy(tekst, zrodlo.tekst);
    return *this;
}
```

```
void main()
{
    C1 obj1(1, "to jest obiekt 1");
    C1 obj2;
    cout << "obj1 = "; obj1.wartosc();
    cout << "obj2 = "; obj2.wartosc();
    obj2 = obj1;
    cout << "obj2 = "; obj2.wartosc();
    obj1.set_tekst("zmiana tekstu");
    cout << "obj2 = "; obj2.wartosc();
}
```

Wynik:

```
obj1 = 1 to jest obiekt 1
obj2 = 0 pusty obiekt
obj2 = 0 to jest obiekt 1
obj2 = 0 to jest obiekt 1
```

Pytanie

Jak odwołać się do metody klasy nadrzędnej?

Funkcje i klasy zaprzyjaźnione

- Funkcje zaprzyjaźnione z klasą to funkcje, które posiadają dostęp do składowych prywatnych
  i zabezpieczonych danej klasy, same nie będąc składowymi tej klasy
- O tym, że dana funkcja jest zaprzyjaźniona z daną klasą informuje deklaracja friend w ciele tej klasy
- Funkcjami zaprzyjaźnionymi mogą być metody innej klasy
- Ta sama funkcja może być zaprzyjaźniona z wieloma klasami

```
class A
{
    int i, j;
    friend void pokaz(A&);
    friend void C::f1();
    friend class B;
};

class B
{
    void zerujA(A &a)
    {
        a.i = a.j = 0;
    }
};

void pokaz(A &a)
{
    cout << a.i << " " << a.j;
}
```

Funkcje i klasy zaprzyjaźnione (2)

```
class Odcinek;

class Punkt
{
    int x;
    char nazwa[10];

public:
    Punkt (int p_x, char *p_nazwa);
    void Przesun (int p_dx) { x += p_dx; };
    friend bool Sprawdz (Punkt &p_p, Odcinek &p_o);
};

Punkt::Punkt (int p_x, char *p_nazwa)
{
    x = p_x;
    strcpy(nazwa, p_nazwa);
}
```

```
class Odcinek
{
    int x, a;
    char nazwa[9];

public:
    Odcinek (int p_x, int p_a, char *p_nazwa);
    friend bool Sprawdz (Punkt &p_p, Odcinek &p_o);
};

Odcinek::Odcinek (int p_x, int p_a, char *p_nazwa)
{
    x = p_x;
    a = p_a;
    strcpy(nazwa, p_nazwa);
}

bool Sprawdz (Punkt &p_p, Odcinek &p_o)
{
    if ( p_p.x >= p_o.x && p_p.x <= p_o.x + p_o.a )
    {
        cout << "p_px=" << p_p.x
            << " p_ox=" << (p_o.x + p_o.a)
            << " na odcinku" << endl;
        return true;
    }
    else
    {
        cout << "poza odcinkiem" << endl;
        return false;
    }
}
```

Funkcje i klasy zaprzyjaźnione (3)

```
class Odcinek;

class Punkt
{
    int x;
    char nazwa[10];

public:
    Punkt (int p_x, char *p_nazwa);
    void Przesun (int p_dx) { x += p_dx; };
    bool Sprawdz (Odcinek &p_o);
};

Punkt::Punkt (int p_x, char *p_nazwa)
{
    x = p_x;
    strcpy(nazwa, p_nazwa);
}
```

```
void main()
{
    Punkt p(2, "punkt");
    Odcinek o(1, 10, "odcinek");
    while (p.Sprawdz(o))
    {
        p1.Przesun(1);
    }
}
```

```
class Odcinek
{
    int x, a;
    char nazwa[9];

public:
    Odcinek (int p_x, int p_a, char *p_nazwa);
    friend bool Punkt::Sprawdz (Odcinek &p_o);
};

Odcinek::Odcinek (int p_x, int p_a, char *p_nazwa)
{
    x = p_x;
    a = p_a;
    strcpy(nazwa, p_nazwa);}

bool Punkt::Sprawdz (Odcinek &p_o)
{
    if ( x >= p_o.x && x <= p_o.x + p_o.a )
    {
        cout << "p_px=" << x
            << " p_ox=" << (p_o.x + p_o.a)
            << " na odcinku" << endl;
        return true;
    }
    else
    {
        cout << "poza odcinkiem" << endl;
        return false;
    }
}
```

Funkcje i klasy zaprzyjaźnione (4)

- W przypadku gdy wszystkie metody jednej klasy mają być zaprzyjaźnione z drugą klasą, można uczynić całą pierwszą klasę klasą zaprzyjaźnioną z drugą
- Brak przechodniości zaprzyjaźnienia
- Zaprzyjaźnienie nie jest dziedziczone

```
class A
{
    //...
    friend class B;
};

class B
{
    //...
    friend class C;
};

class C
{
    //...
};
```

Zadanie

- Napisz klasę Gitara do przechowywania informacji o gitarach
- Napisz klasę Magazyn, która wewnętrznie będzie przechowywać tablicę Gitar
- Przeciąż operatory „+” i „-” tak aby za pomocą tych operatorów było można dodawać i usuwać Gitary z Magazynu
- Do definicji co najmniej jednego operatora wykorzystaj słowo kluczowe friend