



FAKULTA ELEKTROTECHNIKY **ústav**
A KOMUNIKAČNÍCH **teoretické a experimentální**
TECHNOLOGIÍ **elektrotechniky**

Seminář C++

6. přednáška

Autor: doc. Ing. Jan Mikulka, Ph.D.

2023



Obsah přednášky

- Přetěžování funkcí
- Přetěžování operátorů
 - Unární operátory, binární operátory
 - Definice operátorové funkce jako
 - členské metody
 - globální (spřátelené) funkce
 - Přetížení operátoru přiřazení
 - Přetížení operátorů [] a () pro indexování vnitřních polí
 - Přetížení operátoru pro přesměrování objektu na konzoli

Přetěžování funkcí

- V programu se může vyskytovat více funkcí stejného jména s různým počtem parametrů nebo jiného typu.
- V době překladu se funkce přejmenují.
- Při volání se hledá shoda nebo největší pravděpodobnost.

```
int max(int a, int b);  
  
char* max(const char *a, const char *b);  
  
int max(int a, char *b);
```

- Volání funkce `max(2, 3.2)` nenajde shodu, ale najde podobnost v první deklaraci, díky implicitní konverzi typu **double** na typ **int**.

Přetěžování operátorů

- Operátory se definují jako funkce. Přetížení se provede definicí tzv. operátorové funkce s klíčovým slovem **operator**, za kterým následuje znak operátoru a dále parametry operátorové funkce.
- Lze přetěžovat
 - unární, binární operátory,
 - operátory **new** a **delete**,
 - operátory **->**, **[]**, **()**, **=**

Unární operátory

- Unární operátory přetěžíme buď jako členskou funkci třídy bez parametru:
 - typ **operator@()**
- Nebo jako globální funkci s jedním parametrem:
 - typ **operator@(operand)**

Unární operátory, příklad

```
class komplex
{
    double re, im;

public:
    komplex(double re, double im) {
        this->re = re;
        this->im = im;
    }

    void tisk(void) {
        cout << noshowpos << re << showpos << im << "i" << noshowpos;
    }

    komplex operator~() {
        return komplex(re, -im);
    }
};
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    komplex m(2.1, 3.6), n(0, 0);

    n = ~m;

    cout << "Komplexni cislo: ";
    m.tisk();
    cout << endl << "Komplexne sdruzene cislo: ";
    n.tisk();
    cout << endl << endl;

    system("pause");
    return 0;
}
```

Přetížený operátor ~
na funkci komplexně
sdruženého čísla

Inkrementace, dekrementace

- V jazyce C jsou definovány dva typy inkrementace/dekrementace a to:

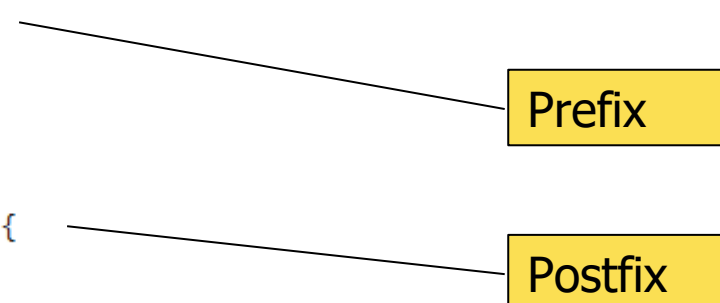
– Prefixová

```
int i = 0;  
cout << ++i; // 1
```

– Postfixová

```
int j = 0;  
cout << j++; // 0
```

```
class komplex  
{  
    double re, im;  
  
public:  
    komplex(double re, double im) {  
        this->re = re;  
        this->im = im;  
    }  
  
    void tisk(void) {  
        cout << noshowpos << re << showpos << im << "i" << noshowpos;  
    }  
  
    komplex operator++() {  
        ++re; ++im;  
        return *this;  
    }  
  
    komplex operator++(int) {  
        komplex k(*this);  
        ++re; ++im;  
        return k;  
    }  
};
```



Binární operátory

- Binární operátory přetěžíme jako členské funkce s jedním parametrem:
 - typ **operator**@(operand)
- Nebo globální funkci se dvěma parametry:
 - typ **operator**@(operand1, operand2)

Binární operátory, příklad

Použití binárního
operátoru +

```
class komplex
{
    double re, im;

public:
    komplex(double re, double im) {
        this->re = re;
        this->im = im;
    }

    void tisk(void) {
        cout << noshowpos << re << showpos << im << "i" << noshowpos;
    }

    komplex operator+(komplex &operand) {
        komplex soucet(this->re + operand.re, this->im + operand.im);
        return soucet;
    }
};
```

Binární operátor

```
int _tmain(int argc, _TCHAR* argv[])
{
    komplex a(2.1, 3.6), b(10, 20), c(0, 0);

    c = a + b;

    cout << "Soucet: ";
    a.tisk();
    cout << " + ";
    b.tisk();
    cout << " = ";
    c.tisk();

    cout << endl << endl;

    system("pause");
    return 0;
}
```

Binární operátory

- Operátorová funkce definovaná jako členská funkce umožňuje binární operace pouze v případě, že levým operandem je instance dané třídy, tj.:

```
komplex a(2.1, 3.6), b(10, 20), c(0, 0);  
c = a + b;
```

- Předpokládejme, že máme implementovaný konverzní konstruktor z typu double. Můžeme součet napsat takto?

```
komplex a(2.1, 3.6), b(10, 20), c(0, 0);  
c = 3.0 + b;
```

```
komplex a(2.1, 3.6), b(10, 20), c(0, 0);  
c = a + 3.0;
```

Binární operátory, definice nečlenskou funkcí

```
class komplex
{
    double re, im;

public:
    komplex(double re, double im) {
        this->re = re; this->im = im;
    }

    komplex(double d) {
        re = d; im = 0;
    }

    void tisk(void) {
        cout << noshowpos << re << showpos << im << "i" << noshowpos;
    }

    /*komplex operator+(komplex &operand) {
        komplex soucet(this->re + operand.re, this->im + operand.im);
        return soucet;
    }*/

    friend komplex operator+(const komplex &operand1, const komplex &operand2);
};

komplex operator+(const komplex &operand1, const komplex &operand2) {
    return komplex(operand1.re + operand2.re, operand1.im + operand2.im);
}
```

Binární operace; definice nečlenskou funkcí

- Pomocí definice operátorové funkce globální, nečlenskou, spřátelenou funkcí dosáhneme toho, že můžeme zaměnit pořadí operandů explicitně/implicitně přetypovaných proměnných na typ daného objektu.
- Důležité prvky v předchozím příkladu:
 - Konverzní konstruktor z typu double na typ komplex.
 - Operátorová funkce jako spřátelená funkce a její konstantní (**const**) parametry. Pokud neuvedeme parametry operátorové funkce jako konstantní, překladač bude hlásit chybu nenalezené operátorové funkce.

Přetěžování binárních operátorů

- Obecná doporučení:
 - Binární operátory přetěžovat jako nečlenské, spřátelené funkce.
 - Parametry definovat jako konstantní (z důvodu překladu) reference (z důvodu šetření času a paměti).
 - Definovat konverzní konstruktory pro všech typy, na jejichž přetypování předpokládáme.

Operátor přiřazení

- Definuje vnitřní přiřazení zapouzdřených proměnných při přiřazení dvou objektů.
- Je třeba myslet na možnost zřetězení přiřazovacího operátoru, př. $A = B = C$;
- Pokud jsou v objektu zapouzdřené pointery, vždy je nutné přetížit operátor přiřazení.

Operátor přiřazení, příklad

```
class pole {  
    int *p, s;  
public:  
    pole() {  
        p = NULL;  
    }  
  
    pole(int i) : s(i) {  
        p = new int[i];  
        for (int i = 0; i < s; i++) p[i] = 0;  
    }  
  
    pole(pole &x) {  
        p = new int[x.s];  
    }  
  
    pole& operator=(const pole &x) {  
        if (p != NULL) delete [] p;  
        p = new int[s = x.s];  
        for (int i = 0; i < s; i++) p[i] = x.p[i];  
        return *this;  
    }  
  
    ~pole() {  
        delete [] p;  
    }  
};
```

Pokud již má objekt alokované místo pro pole, je třeba ho dealokovat

Následně alokujeme paměť přesně podle velikosti vzorového pole

A můžeme kopírovat data...

Operátory [] a ()

- Operátorům [] a () lze opět definovat jakoukoliv funkci, ale v analogii s jejich významem v jazyce C se užívají hlavně pro
 - indexování zapouzďřených polí v případě [] a
 - v případě () se jedná o význam volání funkce.
- Typickým příkladem může být indexování prvku v objektu Pole nebo prvku Matice; může předávat znak na určité pozici v souboru objektu File apod.
- Operátory závorek lze „řetěžit“, tzn. Můžeme indexovat 1D pole, ale také 2D pole, 3D pole a to opět v analogii s C-přístupem:
 - A[i] ... vrátí *i*. prvek v 1D poli
 - A[i][j] ... vrátí prvek na *i*. řádku a *j*. sloupci ve 2D poli
 - A[i][j][k] ... vrátí prvek na *i*. řádku, *j*. sloupci v hloubce *k* ve 3D poli

Operátor [], příklad

- Příklad přetížení pro 1D pole:

```
int& operator[](int i) {  
    return p[i];  
}
```

```
pole a(10), b(10);  
  
a[2] = 2;  
  
cout << a[2];
```

- Příklad přetížení pro 2D pole:

```
double* operator[](int i) const {  
    return mat[i];  
}
```

```
Matice C(5, 5);  
  
C[1][2] = 3;  
  
cout << C[1][2];
```

Operátor (), příklad

- Příklad použití pro 1D pole ve funkci výběru části pole:

```
pole operator()(int start, int stop) {  
    pole x(stop-start+1);  
    for (int i = start; i <= stop; i++) x.p[i-start] = p[i];  
    return x;  
}
```

```
pole a(10), b(3);  
  
b = a(5, 7);  
  
cout << a << endl << b << endl;
```


Operátory << a >>

- Operátorům lze opět definovat jakoukoliv funkci. Analogicky se předpokládá následující význam:
 - Přesměrování objektu na konzoli (výpis)
 - Posun zapouzdřených dat vlevo/vpravo (v C jsou tyto operátory použity pro bitový posun)

Operátor <<, příklad

- Příklad přesměrování objektu na konzoli – výpis zapouzdřených dat:

```
ostream& operator<<(ostream &out, pole &p) {  
    for (int i = 0; i < p.gets(); i++) out << p.p[i];  
    return out;  
}
```

```
pole a(10), b(3);  
  
cout << a << endl << b << endl;
```

**Děkuji Vám
za pozornost.**

mikulka@vut.cz
www.utee.fekt.vut.cz