Přetížení operátorů (operator overloading)

Přetížení operátorů je definování jejich chování pro další datové typy. Je specifickým typem polymorfismu. Přetížení operátorů je v C++ u některých operátorů definováno v programovacím jazyce (operátory **<<** a **>>** ve streamech) a rovněž uživatel může definovat své vlastní přetížení operátorů.

U přetížených operátorů zůstávají zachovány jejich charakteristické vlastnosti

- arita
- precedence
- asociativita

V definici přetížení musí aspoň jeden z typů operandů být třída.

Lze přetížit operátory:

`+`	`-`	`*`	`/`	`%`	`^`
`&`	`|`	`~`	`!`	`,`	`=`
`<`	`>`	`<=`	`>=`	`++`	`--`
`<<`	`>>`	`==`	`!=`	`&&`	`||`
`+=`	`-=`	`/=`	`%=`	`^=`	`&=`
`|=`	`*=`	`<<=`	`>>=`	`[]`	`()`
`->`	`->*`	`new`	`new []`	`delete`	`delete []`

Operátory **+**, **-**, *****, **&** lze přetížit binární i unární.

Nelze přetížit operátory:

`::`	`.*`	`.`	`?:`	`sizeof`

Přetížení lze definovat:

- Členskou funkcí třídy pro unární operátor nebo pro binární operátor, je-li jeho první (levý) operand třída.
- Globální funkcí.

Tvar členské funkce pro unární operátor

```
typ operator operátor () { }
```

Tvar členské funkce pro binární operátor

```
typ operator operátor (pravý_operand) { }
```

Tvar globální funkce pro unární operátor

```
typ operator operátor (operand) { }
```

Tvar globální funkce pro binární operátor

```
typ operator operátor (levý_operand,pravý_operand) { }
```

Úloha: Máme třídu pro uložení data

```
class Datum { char den,mesic; short rok; };
```

Potřebujeme setřídit pole dat. Použijeme k tomu algoritmus třídění přímým vkládáním, který srovnává tříděné prvky operací srovnání **<**. Pro třídění definujeme tuto operaci ve třídě *Datum*.

```
class Datum { char den,mesic; short rok;
  public: bool operator < (const Datum &d)
          {
            if (rok!=d.rok)      return rok<d.rok;
            if (mesic!=d.mesic) return mesic<d.mesic;
                                 return den<d.den;
          }
};

void tridit(Datum a[], int n)
{
  for (int i=1; i<n; ++i)
  { Datum x=a[i];
    int j;
    for (j=i-1; j>=0 && x<a[j]; --j) a[j+1] = a[j];
    a[j+1] = x;
  }
};
```

Tvar funkce pro konverzi datového typu dané třídy na stanovený datový typ:

```
operator typ () { }
```

```
Retez << ' '
```

```
Retez += " "
```

```
Retez += Retez
```

```
Retez = " "
```

```
Retez = Retez
```

```
(const char *) Retez
```

```
class Retez { char *p; int ix;
  public: Retez(int n) { p=new char[n+1]; ix=0; }
          Retez & operator << (char);
```

```
            Retez & operator += (const char *);
            Retez & operator += (const Retez &);
            Retez & operator = (const char *);
            Retez & operator = (const Retez &);
            operator const char * () const { return p; }
            ~Retez() { delete [] p; }
};

Retez & Retez::operator << (char z)
{
  p[ix++]=z;
  p[ix]=0;
  return *this;
}

Retez & Retez::operator += (const char *s)
{
  strcpy(p+ix,s);
  ix+=strlen(s);
  return *this;
}

Retez & Retez::operator += (const Retez &r)
{
  strcpy(p+ix,r.p);
  ix+=strlen(r.p);
  return *this;
}

Retez & Retez::operator = (const char *s)
{
  strcpy(p,s);
  ix=strlen(s);
  return *this;
}

Retez & Retez::operator = (const Retez &r)
{
  strcpy(p,r.p);
  ix=strlen(r.p);
  return *this;
}

Retez r(100);

r << 'C' << '+' << '+';

r += " je ";
```

```
Retez s(100);

s = "programovaci jazyk";

r += s;

cout << r << endl;   // C++ je programovaci jazyk
```

Přetížení operátorů ++ a --

Přetížení prefixových operátorů **++** a **--** má tvar:

```
  typ operator operátor () { }
```

Funkce pro přetížení postfixových operátorů **++** a **--** má tvar:

```
  typ operator operátor (int) { }
```

```
class Souradnice { float x, krok;
  public:
    Souradnice(float x,float k):x(x) { krok=k; }
    float operator ++ () { return x+=krok; }
    float operator ++ (int) { float v=x; x+=krok; return v; }
    operator float () const { return x; }
};

Souradnice s(0,2);

cout << ++s << endl;  // 2

cout << s++ << endl;  // 2

cout << s << endl;    // 4
```

Přetížení operátoru indexu

```
  typ operator [] (index) { }
```

```
class Pole { const unsigned n;
             int *p;
  public: Pole(unsigned n):n(n) { p=new int [n+1]; }
          int & operator [] (unsigned i) const
          {
            if (i<n) return p[i];
            cerr << "Chybny index";
            return p[n];
          }
```

```
        ~Pole() { delete [] p; }
};
```

Přetížení operátoru funkce

```
  typ operator () (…argumenty…) { }
```

```
class Rovnice { float a,b;
  public: void operator () (float aa,float bb) { a=aa; b=bb; }
          float operator () () const { return -b/a;}
};

Rovnice r;

r(4,-3);

cout << r();    // 0.75
```

Operátor přiřazení - hluboké kopírování

```
struct Jmeno { char *r;
  Jmeno() { r=new char [21]; }
  Jmeno & operator = (const char *j)
  {
    strcpy(r,j); return *this;
  }
  ~Jmeno() { delete [] r; }
};

Jmeno e,j;

e="Eva";

j=e;

j="Jana";

cout << e.r << endl;    // Jana
cout << j.r << endl;    // Jana

struct Jmeno { char *r;
    Jmeno() { r=new char [21]; }
    Jmeno & operator = (const char *j)
    {
      strcpy(r,j); return *this;
    }
```

```
    Jmeno & operator = (const Jmeno &j)
    {
      strcpy(r,j.r); return *this;
    }
    ~Jmeno() { delete [] r; }
};

cout << e.r << endl;     // Eva
cout << j.r << endl;     // Jana
```

Přetížení binárních aritmetických operátorů – příklad

```
class Zlomek { unsigned c,j;
          void nsd()
          { if (c!=0) { unsigned a=c,b=j;
              for (;;) { unsigned r=a%b;
                if (r==0) { if (b>1) c/=b,j/=b; return; }
                a=b; b=r; } }
            j=1; }
  public: Zlomek() { }
          Zlomek(unsigned c, unsigned j):c(c),j(j) { nsd(); }
          Zlomek operator * (const Zlomek &z) const
          {
            return Zlomek(c*z.c, j*z.j);
          }
          unsigned cit() const { return c; }
          unsigned jm() const { return j; }
          void operator () (const char *s="\n") const
          {
            cout << c << '/' << j << s;
          }
};

Zlomek a(2,3),b(1,2);

Zlomek &z = a*b*Zlomek(7,3);

z();    // 7/9
```

Přetížení operátoru new – příklad

```
class Uzel { int hodnota; Uzel *levy,*pravy;
  public:
```

```
    static void * pamet(size_t s)
    {
      static size_t v=0;
      static char *p;
      if (v<s) { size_t w=v=1000*sizeof(Uzel);
                 if (w<s) return malloc(s);
                 p=(char *)malloc(v=w); }
      char *q=p; p+=s; v-=s;
      return q;
    }
    void * operator new (size_t s) { return pamet(s); }
    void * operator new [] (size_t s) { return pamet(s); }
  private:
    void operator delete (void *) { }
    void operator delete [] (void *) { }
};
Uzel *u=new Uzel;
delete u;     // nelze – operátor delete je soukromý
```