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) { j=1; return; }
```

```
            unsigned a=c,b=j,r;
            for (;;) { r=a%b; if (r==0) { c/=b; j/=b; return; }
                       a=b; b=r; }
          }
  public: Zlomek() { }
          Zlomek(unsigned c, unsigned j):c(c),j(j) { nsd(); }
          Zlomek operator * (const Zlomek &z)
          {
            return Zlomek(c*z.c, j*z.j);
          }
          void operator () () const
          {
            cout << c << '/' << j << endl;
          }
};

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

Zlomek &z = a*b*c;

z();    // 1/4
```

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 – operator delete je soukromy
```