13 Operaattoreiden ylimäärittelyjä

C++-kielessä voidaan operaattoreita ylimäärittää. Ylimääriteltävää operaattoria voidaan pitää ikäänkuin metodina, joka esitellään luokan esittelyssä ja määritellään luokan ulkopuolella kuten metoditkin. Ylimääritellyillä operaattoreilla on kuitenkin poikkeavat nimet ja vain olemassaolevia operaattoreita voidaan ylimääritellä.

C++-kielessä käytetään operaattoreiden ylimäärittelyjä muutoinkin: esimerkiksi yhteenlaskuoperaattori (+) on ylimääritelty laskemaan yhteen kokonaislukuja ja liukulukuja, vaikka niiden talletustavat ovat erilaiset.

Sisäisten tietotyyppien operaattoreita ei voida ylimääritellä.

Muissa kielissä saatetaan ylimäärittelyssä mennä vieläkin pitemmälle: esimerkiksi Java-kielessä voidaan +-operaattorilla yhdistää merkkijonoja.

C++-kielessä voidaan ylimääritellä seuraavia operaattoreita:

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

Esittelyssä ja määrittelyssä käytetään avainsanaa operator, jota seuraa ylimääriteltävä operaattori.

Operaattorin ylimäärittely voidaan toteuttaa metodin tai ystävä-funktion kautta.

Binääreillä operaattoreilla on aina 1 parametri ja unaareilla ei ole parametria lainkaan silloin, kun operaattorit on ylimääritelty luokan jäseninä.

13.1 Operaattorin + ylimäärittely

Seuraavassa ylimääritellään yhteenlaskuoperaattori laskemaan yhteen kaksi kompleksilukua. Jäsenmuuttujat vastaavat kompleksiluvun reaali- ja imaginääriosia.

Ylimäärittelyn toteuttavan funktion määrittely on seuraavanlainen:

operator+

```
complex operator+ (const complex a, const complex b)
{
complex temp(0,0);
temp.x = b.x + a.x;
temp.y = b.y + a.y;
return temp;
}
```

Parametreina on kaksi vakiota complex-oliota ja palautusarvona on olio, jonka jäsenmuuttujina ovat parametreina vietyjen olioiden jäsenmuuttujien summat.

Nyt yhteenlaskuoperaattorin operandeina voi olla olioita:

```
complex c = a + b;
```

Ylimääritelty operator+ ohjelmassa:

```
#include <iostream.h>

class complex {
    double x, y;
public:
    complex (double real, double imag) { x=real; y=imag; }
    friend complex operator+ (const complex a, const complex b);
    friend void main();
    };


complex operator+ (const complex a, const complex b)
{
complex temp(0,0);
temp.x = b.x + a.x;
temp.y = b.y + a.y;
return temp;
}


void main()
{
complex a(2,3);
cout << a.x << a.y << "\n";
complex b(1,1);
complex c = a + b;
cout << a.x << a.y << "\n";
```

```
cout << b.x << b.y << "\n";
cout << c.x << c.y << "\n";
}
```

Kun operaattori esitellään luokan metodina, on yleensä operaattorilla vain yksi parametri.
Seuraavassa esimerkissä käytetään yhteenlaskuoperaattoria luokan jäsenenä.

operator+ ohjelmassa luokan jäsenenä:

```
#include <iostream.h>
typedef unsigned short USHORT;

class Laskuri
{
public:
  Laskuri ();
  ~ Laskuri (){}
  USHORT HaeArvo()const { return arvo; }
  void AsetaArvo(USHORT x) {arvo = x; }
  Laskuri operator+ (const Laskuri &);

private:
  USHORT arvo;

};

Laskuri:: Laskuri ():
arvo(0)
{};

Laskuri & Laskuri::operator+(const Laskuri & oikeapuoli)
{
  return Laskuri(arvo + oikeapuoli.HaeArvo());
}


int main()
{
 Laskuri L;
 Laskuri eka, toka, kolkku;
 eka.AsetaArvo(4);
 toka.AsetaArvo(5);
 kolkku = eka + toka;
 cout << kolkku.HaeArvo();
 return 0;
}
```

Seuraavassa ohjelmassa on ylimääritelty +-operaattori muuttamaan penneinä annettu rahasumma markoiksi ja penneiksi. Ohjelmassa on myös erikoismuodostin, joka muuntaa parametrinä viedyn pennisumman olioksi, jonka jäsenmuuttujiin sijoitetaan vastaavat markka- ja pennimäärät.

operator+ ja muunnosmuodostin:

```
#include <iostream.h>

class Raha {
public:
     int markat;
     int pennit;
public:
     Raha() {}
     ~Raha() {}
     Raha(int);                 // muunnos double-tyypistä
     Raha operator+(Raha R);     // operaattorin uudelleenmäärittely
};

Raha Raha::operator+(Raha R)
{
     pennit += R.pennit;
     markat += R.markat;

     markat = pennit / 100;
     pennit = pennit % 100;

     return *this;
}

Raha::Raha(int pennisumma) // muunnosmuodostin muuntaa int-tyypin
Raha-olioksi
{
     markat = pennisumma / 100;
     pennit = pennisumma % 100;
}

int main()
{
     int a, b;
     int pennit1 = 550;
     int pennit2 = 220;

     Raha lompakko = pennit1 + pennit2;
```

```
        a = lompakko.markat;
        b = lompakko.pennit;

        cout << "Lompsassasi on nyt " << a << " markkaa\n";
        cout << "Sekä " << b << " penniä\n";

        return 0;
    }
```

Seuraavana on ylimääriteltynä operaattori +=, joka esitellään luokkaan kuuluvana operaattorina:

operator +=

```
complex complex:: operator+= (const complex p)
{
x += p.x;
y += p.y;
return *this;
}
```

operator += ohjelmassa:

```
#include <iostream.h>

class complex {
    double x, y;
public:
    complex (double real, double imag) { x=real; y=imag; }
    complex operator+= (const complex p);
    friend void main();
    };

complex complex:: operator+= (const complex p)
{
x += p.x;
y += p.y;
return *this;
}


void main()
{
complex a(2,3);
```

```
cout << a.x << a.y << "\n";
complex b(1,1);

a += b;

cout << a.x << a.y << "\n";
cout << b.x << b.y << "\n";
}
```

13.2 Jakolaskuoperaattori (/)

Seuraavana on ohjelma, jossa Murtoluku-luokan jäsenmuuttujina ovat jakaja ja jaettava ja ylimääritelty jakolaskuoperaattori (/) on määritelty toteuttamaan kahden murtolukuolion jakolasku:

operator/

```
#include <iostream.h>

class Murtoluku
{
int jakaja;
int jaettava;
public:
Murtoluku();
Murtoluku(int a, int b){jakaja = b;jaettava = a;}

friend main();
friend Murtoluku operator /(Murtoluku& c1, Murtoluku& c2);
friend void Sievenna(Murtoluku & c1);
};

Murtoluku::Murtoluku()
{
jakaja = 0;
jaettava = 0;
}

void Sievenna(Murtoluku & c1)
{
int syt;
int a = c1.jakaja;
int b = c1.jaettava;
      while (a != b)
            {  if (a > b) a = a-b;
```

```
                else b = b-a;
                syt = a;
              }
    c1.jakaja = c1.jakaja/syt;
    c1.jaettava = c1.jaettava/syt;
}

Murtoluku operator /(Murtoluku& c1, Murtoluku& c2)
{
Murtoluku tulos;
tulos.jakaja = c1.jaettava * c2.jakaja;
tulos.jaettava = c2.jaettava * c1.jakaja;
return tulos;
}


main()
{
int p;
Murtoluku Luku1(2,3);
Murtoluku Luku2(5,6);
Murtoluku Luku3 = Luku1/Luku2; // 2/3 : 5/6  = 2/3 * 6/5 = 12/15
cout << "3. murtoluvun jakaja on " << Luku3.jakaja << "\n";
cout << "3. murtoluvun jaettava on " << Luku3.jaettava << "\n";

Sievenna(Luku3);
cout << "3. murtoluvun jakaja on " << Luku3.jakaja << "\n";
cout << "3. murtoluvun jaettava on " << Luku3.jaettava << "\n";


return 0;
}
```

Sijoitusoperaattori (=) tulee ylimääritellä esimerkiksi silloin, kun jäsenmuuttujina on osoittimia. Tällöin sijoitusoperaattorin tulee huolehtia siitä, että osoittimissa olevat muistiosoitteet ovat uniikkeja.

Sijoitusoperaattori samoin kuin hakasulkuoperaattorikin tulee esitellä luokan jäsenenä. Myös ns. funktion kutsu -operaattori () tulee esitellä jäsenenä.

13.3 operator++

Kasvatusoperaattori voidaan myöskin ylimääritellä toimimaan halutulla tavalla. Seuraavassa ohjelmassa luokan jäseneksi määritelty kasvatusoperaattori kasvattaa jäsenmuuttujan arvoa.

Operaattorin toteutus on selkeä:

```
Laskuri & Laskuri::operator++()
{
  ++arvo;
  return *this;
}
```

Kun toteutus palauttaa this-osoittimen, voidaan sitä hyödyntää myös sijoituksessa.

Seuraavana on kokonainen ohjelma, joka sisältää myös metodin Kasvata().

operator++ ohjelmassa:

```
#include <iostream.h>
typedef unsigned short  USHORT;

class Laskuri
{
public:
 Laskuri ();
  ~ Laskuri (){}
  USHORT HaeArvo()const { return arvo; }
  void AsetaArvo(USHORT x) {arvo = x; }
  void Kasvata() { ++arvo; }
  Laskuri& operator++ ();

private:
  USHORT arvo;

};

Laskuri:: Laskuri ():
arvo(0)
{};

Laskuri & Laskuri::operator++()
{
  ++arvo;
  return *this;
}

int main()
{
 Laskuri L;
  cout << "L:n jäsenmuuttujan arvo on " << L.HaeArvo() << endl;
  L.Kasvata();
```

```
  cout << " L:n jäsenmuuttujan arvo on " << L.HaeArvo() << endl;
  ++L;
  cout << " L:n jäsenmuuttujan arvo on " << L.HaeArvo() << endl;
  Laskuri K = ++L;
  cout << " K:n jäsenmuuttujan arvo on " << K.HaeArvo() << endl;
  cout << " ja L:n: " << L.HaeArvo() << endl;
  return 0;
}
```

13.4 operator unsigned short()

C++ sisältää muunnosoperaattoreita, joita voidaan lisätä luokkaan. Luokka voi määrittää, kuinka tehdä muunnokset sisäisiin tyyppeihin. Muunnosoperaattorit eivät määritä palautusarvoa, vaikka ne palauttavatkin muunnetun arvon.

Seuraavassa esimerkissä unsigned short() muuntaa arvon int-tyypiksi.

unsigned short() ohjelmassa:

```
#include <iostream.h>
typedef unsigned short USHORT;
class Laskuri
{
public:
    Laskuri();
    Laskuri(USHORT x);
~Laskuri(){}
USHORT HaeArvo()const { return arvo; }
void AsetaArvo(USHORT x) {arvo = x; }
operator unsigned short();
private:
USHORT arvo;

};

Laskuri::Laskuri():
arvo(0)
{}

Laskuri::Laskuri(USHORT x):
arvo(x)
{}

Laskuri::operator unsigned short()
```

```
{
  return (int (arvo) );
}

int main()
{
     Laskuri L1(5);
     int luku = L1;
     cout << "Muuttujan luku arvo on: " << luku << endl;
     L1.AsetaArvo(10);
     luku = L1;
     cout << "Muuttujan luku arvo on nyt: " << luku << endl;
     return 0;
}
```

13.5 operator[]

Hakasulkuoperaattorin ylimäärittely saattaa olla useinkin hyödyllinen. Voimme tällöin esimerkiksi tarkistaa, ettei taulukon rajoja ylitetä.

Seuraavana on ohjelma, joka tekee indeksien tarkistuksen.

operator[] ohjelmassa:

```
#include <iostream.h>

#define koko 10

class taulu{
         int indeksi;
         unsigned jono[koko];
     public:
         taulu(unsigned arvo = 0);
         unsigned & operator[](int indeksi);
};

taulu::taulu(unsigned arvo) {
     for (int i = 0; i < koko; i++)
         jono[i] = arvo;
};

unsigned & taulu::operator[](int indeksi) {
     if (indeksi < 0 )
         cout << "Indeksin on oltava positiivinen\n";
     if (indeksi >= koko)
         cout << "Indeksi ylittää maksimikoon " << koko << "\n";
```

```
    return jono[indeksi];
};


int main()
{
    taulu a(0);  // taulukko, jossa 10 alkiota, alustetaan nollalla
    a[0] = 5;        // Oikein!
    a[100] = 10;// Virheilmoitus
    a[-1] = 15;  // Virheilmoitus
    return 0;
}
```

13. 6 Syötön (>>) ja tulostuksen (<<) ylimäärittely

Myös syötön ja tulostuksen toteutumista voidaan muuttaa. Tulostusvirtaluokka on ostream ja syöttövirtaluokka istream. Voimme siis esitellä oliot, jotka ovat noita luokkatyyppejä ja käsitellä tulostus- ja syöttövirtoja noiden olioiden avulla. Luokat ostream ja istream on määritelty stream-kirjastossa.

Seuraavana on ohjelma, jossa on tulostusoperaattori ylimääritelty. Se tulostaa muutettuna olion jäsenmuuttujan arvon. Operaattorin toteutukseen viedään parametreina tulostusolio sekä ohjelmassa määritelty olio (viittaukset niihin).

Tulostuksen ylimäärittely

```
#include <iostream.h>

class HLO {
public:
    friend ostream& operator<< (ostream& out, const HLO& hlo);
    HLO(int x) {hlonro = x;}
private:
    int hlonro;
};

ostream& operator<< (ostream& out, const HLO& hlo)
{
  return out << hlo.hlonro;
}

main()
```

```
{
    HLO Veikko(100);
    cout << "HLO-olio: " << Veikko << "\n";
}
```

13.7 operator <

Seuraavassa esimerkissä on operaattori < ylimääritelty vertaamaan 2 merkkijonoa keskenään:

operator <

```
#include <iostream.h>
#include <string.h>

class String {
          char *mj;
public:
    String(){mj = "KAUKO";}
    String( char * );
    ~String() { delete mj; }
    friend int operator < (String, String);
    friend void main();
};


String::String(char *s)
{
    mj = new char[strlen(s) + 1];
    strcpy(mj, s);
}

int operator < (String s1, String s2)
{
         return strcmp (s1.mj, s2.mj);
}

void main()
{
          String s0 ;
          String s1 ("Juhuu");
          String s2 ("Heippati");

          cout << s1.mj << "\n";
          cout << s2.mj << "\n";
```

```
            cout << (s1 < s0) << "\n";
            cout << (s1 < s2) << "\n";
    }
```

13.8 operator()

Operator() täytyy esitellä luokan jäsenenä. Sen erikoisuus on siinä, että se voidaan määritellä ottamaan useita parametreja toisin kuin muut luokan jäseninä olevat operaattorit.

Seuraavana on esimerkki, jossa operator() määritellään ottamaan matriisin rivi- ja sarakemäärät parametreikseen sekä alustamaan sitten luokan tietojäsenenä olevan 2-ulotteisen taulukon. Hakasulkuoperaattoria ei voida määritellä ottamaan useampia parametreja.

operator()

```
#include <iostream.h>
#include <assert.h>
#include <stdlib.h>

const int rivit= 4;
const int sarakkeet= 3;

class taulu {
    int data[rivit][sarakkeet];
public:
    int& operator() (int x, int y);
    friend void main();
    };

int& taulu::operator() (int x, int y)
{
assert (x >= 0 && y >= 0 && x <= rivit && y <= sarakkeet);
return data[x][y];
}

void main()
{
int a, b, i, j;
cout << "Anna rivien määrä \n";
cin >> a;
cout << "Anna sarakkeiden määrä \n";
cin >> b;
```

```
taulu t1;
cout << "Täytetään \n";
for (i = 0; i < a; i++)
for (j = 0; j < b; j++)
t1(i,j) = rand() % 9;

cout << "TULOSTETAAN \n";
for (i = 0; i < a; i++)
for (j = 0; j < b; j++)
cout << t1(i,j) << "\n";
}
```

Muut operaattorit

operator& voidaan ylimääritellä. Vaarana on tällöin se, että osoitteeseen ei päästäkään enää ylimäärittelyn jälkeen käsiksi. &-operaattori on unaarinen (eli yksioperandinen) operaattori. Sen ylimäärittelyminen on harvinaista juuri sen takia, että sitä käytetään normaalistikin luokan toiminnoissa.
Pilkkuoperaattori (,) on myöskin harvoin ylimääritelty operaattori. Sitäkin käytetään normaalisti oliopohjaisissa ohjelmissa.