```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  Afmeting.h
// Implemen: Afmeting.cp
//=======================================

#pragma once

c_n4 aRij=6;
c_n4 aKolom=7;
c_n4 lNiveau=aRij*aKolom;
c_n4 aNiveau=lNiveau+1;

#define BORDBIT(rij,kolom) (1ULL<<((rij)+((kolom)<<3)))
```

```
//=========================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  Afmeting.cp
//=========================================

#include "Afmeting.h"
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  Uitslag.h
// Implemen: Uitslag.cp
//=======================================

#pragma once

// opm: geel gaat beginnen in kolom 3.
// Geel gaat winnen,
// dus gelijkspel zal nooit voorkomen.
// Daarom wordt gelijkspel gelijkgesteld
// met winst voor rood.

enum Uitslag
{
    geelWint,
    roodWint,
    onbekend
};

    typedef Uitslag* p_Uitslag;
    typedef const p_Uitslag cp_Uitslag;
typedef const Uitslag c_Uitslag;
    typedef c_Uitslag* pc_Uitslag;
    typedef const pc_Uitslag cpc_Uitslag;
typedef Uitslag& r_Uitslag;
typedef p_Uitslag& rp_Uitslag;
typedef cp_Uitslag& rcp_Uitslag;
typedef c_Uitslag& rc_Uitslag;
typedef pc_Uitslag& rpc_Uitslag;
typedef cpc_Uitslag& rcpc_Uitslag;
```

```
//=========================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  Uitslag.cp
//=========================================

#include "Uitslag.h"
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  Raat.h
// Implemen: Raat.cp
//=======================================

#pragma once

union Raat;
    typedef Raat* p_Raat;
    typedef const p_Raat cp_Raat;
typedef const Raat c_Raat;
    typedef c_Raat* pc_Raat;
    typedef const pc_Raat cpc_Raat;
typedef Raat& r_Raat;
typedef p_Raat& rp_Raat;
typedef cp_Raat& rcp_Raat;
typedef c_Raat& rc_Raat;
typedef pc_Raat& rpc_Raat;
typedef cpc_Raat& rcpc_Raat;

// opm: big endian.

union Raat
{
    n8 eenDeel;
    n1 tabel[8];
    struct
        {
        n1 leeg;
        n1 kolom6;
        n1 kolom5;
        n1 kolom4;
        n1 kolom3;
        n1 kolom2;
        n1 kolom1;
        n1 kolom0;
        } achtDelen;
};
```

```
//=========================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  Raat.cp
//=========================================

#include "Raat.h"
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  Bord.h
// Implemen: Bord.cp
//=======================================

#pragma once

#include "Raat.h"

class Bord;
    typedef Bord* p_Bord;
    typedef const p_Bord cp_Bord;
typedef const Bord c_Bord;
    typedef c_Bord* pc_Bord;
    typedef const pc_Bord cpc_Bord;
typedef Bord& r_Bord;
typedef p_Bord& rp_Bord;
typedef cp_Bord& rcp_Bord;
typedef c_Bord& rc_Bord;
typedef pc_Bord& rpc_Bord;
typedef cpc_Bord& rcpc_Bord;

class Bord
{
protected:
    // Eigen data.
    Raat m_geel;
    Raat m_rood;

    // Constructie / Kopie / Destructie.
public:
    Bord();
    Bord(n8 geel,
         n8 rood);
    Bord(rc_Raat geel,
         rc_Raat rood);
    Bord(rc_Bord origineel);
    ~Bord();

    // Operators.
    bool operator==(rc_Bord x) const;
    bool operator!=(rc_Bord x) const;
```

```
    bool operator>(rc_Bord x) const;
    bool operator<(rc_Bord x) const;
    bool operator>=(rc_Bord x) const;
    bool operator<=(rc_Bord x) const;
    void operator=(rc_Bord origineel);

    // Manipulators.
    Orde Vergelijk(rc_Bord bord) const;
    bool IsSymmetrisch() const;
    void Geef_geel(r_n8 geel) const;
    void Geef_rood(r_n8 rood) const;
    bool GeelWint() const;
    bool RoodWint() const;
private:
    bool Wint(n8 kleur) const;
public:
    bool Bevat(rc_Bord bord) const;
    void Spiegel();
private:
    void Verwissel(r_n1 x,
                   r_n1 y) const;
};
```

```
//=========================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  Bord.cp
//=========================================

#include "Bord.h"

#include "Afmeting.h"

#pragma profile off

Bord::Bord()
{
    m_geel.eenDeel=0;
    m_rood.eenDeel=0;
}

Bord::Bord(c_n8 geel,
           c_n8 rood)
{
    m_geel.eenDeel=geel;
    m_rood.eenDeel=rood;
}

Bord::Bord(rc_Raat geel,
           rc_Raat rood)
{
    m_geel.eenDeel=geel.eenDeel;
    m_rood.eenDeel=rood.eenDeel;
}

Bord::Bord(rc_Bord origineel)
{
    m_geel.eenDeel=origineel.m_geel.eenDeel;
    m_rood.eenDeel=origineel.m_rood.eenDeel;
}

Bord::~Bord()
{
}

void Bord::operator=(rc_Bord origineel)
{
```

```
    m_geel.eenDeel=origineel.m_geel.eenDeel;
    m_rood.eenDeel=origineel.m_rood.eenDeel;
}

bool Bord::operator==(rc_Bord origineel) const
{
    if (m_geel.eenDeel!=origineel.m_geel.eenDeel)
        {
        return false;
        }
    if (m_rood.eenDeel!=origineel.m_rood.eenDeel)
        {
        return false;
        }
    return true;
}

bool Bord::operator!=(rc_Bord origineel) const
{
    if (m_geel.eenDeel!=origineel.m_geel.eenDeel)
        {
        return true;
        }
    if (m_rood.eenDeel!=origineel.m_rood.eenDeel)
        {
        return true;
        }
    return false;
}

bool Bord::operator>(rc_Bord x) const
{
    c_Orde o=Vergelijk(x);
    c_bool ok=(o==orde_eersteKomtNaTweede);
    return ok;
}

bool Bord::operator<(rc_Bord x) const
{
    c_Orde o=Vergelijk(x);
    c_bool ok=(o==orde_eersteKomtVoorTweede);
    return ok;
}

bool Bord::operator>=(rc_Bord x) const
```

```
{
    c_Orde o=Vergelijk(x);
    c_bool ok=(o!=orde_eersteKomtVoorTweede);
    return ok;
}

bool Bord::operator<=(rc_Bord x) const
{
    c_Orde o=Vergelijk(x);
    c_bool ok=(o!=orde_eersteKomtNaTweede);
    return ok;
}

Orde Bord::Vergelijk(rc_Bord bord) const
{
    if (m_geel.eenDeel<bord.m_geel.eenDeel)
        {
        return orde_eersteKomtVoorTweede;
        }
    if (m_geel.eenDeel>bord.m_geel.eenDeel)
        {
        return orde_eersteKomtNaTweede;
        }
    if (m_rood.eenDeel<bord.m_rood.eenDeel)
        {
        return orde_eersteKomtVoorTweede;
        }
    if (m_rood.eenDeel>bord.m_rood.eenDeel)
        {
        return orde_eersteKomtNaTweede;
        }
    return orde_eersteIsGelijkAanTweede;
}

bool Bord::IsSymmetrisch() const
{
    if (m_geel.achtDelen.kolom2!=m_geel.achtDelen.kolom4)
        {
        return false;
        }
    if (m_geel.achtDelen.kolom1!=m_geel.achtDelen.kolom5)
        {
        return false;
        }
    if (m_geel.achtDelen.kolom0!=m_geel.achtDelen.kolom6)
```

```
        {
        return false;
        }
    if (m_rood.achtDelen.kolom2!=m_rood.achtDelen.kolom4)
        {
        return false;
        }
    if (m_rood.achtDelen.kolom1!=m_rood.achtDelen.kolom5)
        {
        return false;
        }
    if (m_rood.achtDelen.kolom0!=m_rood.achtDelen.kolom6)
        {
        return false;
        }
    return true;
}

void Bord::Geef_geel(r_n8 geel) const
{
    geel=m_geel.eenDeel;
}

void Bord::Geef_rood(r_n8 rood) const
{
    rood=m_rood.eenDeel;
}

bool Bord::GeelWint() const
{
    c_bool wint=Wint(m_geel.eenDeel);
    return wint;
}

bool Bord::RoodWint() const
{
    c_bool wint=Wint(m_rood.eenDeel);
    return wint;
}

bool Bord::Wint(c_n8 kleur) const
{
    // opm: ••••.
    n8 x=kleur bitand (kleur<<2);
    x and_eq (x<<1);
```

```
    if (x)
        {
        return true;
        }

    // opm: •
    //      •
    //      •
    //      •
    x=kleur bitand (kleur<<14);
    x and_eq (x<<7);
    if (x)
        {
        return true;
        }

    // opm:    •
    //        •
    //       •
    //      •
    x=kleur bitand (kleur<<18);
    x and_eq (x<<9);
    if (x)
        {
        return true;
        }

    // opm: •
    //       •
    //        •
    //         •
    x=kleur bitand (kleur<<16);
    x and_eq (x<<8);
    if (x)
        {
        return true;
        }

    return false;
}

bool Bord::Bevat(rc_Bord bord) const
{
    for (z4 i=1;i<8;i++)
        {
```

```
        n1 kolom=bord.m_geel.tabel[i];
        if (m_geel.tabel[i] bitand kolom!=kolom)
            {
            return false;
            }
        kolom=bord.m_rood.tabel[i];
        if (m_rood.tabel[i] bitand kolom!=kolom)
            {
            return false;
            }
        }
    return true;
}

void Bord::Spiegel()
{
    Verwissel(m_geel.achtDelen.kolom2,
              m_geel.achtDelen.kolom4);
    Verwissel(m_geel.achtDelen.kolom1,
              m_geel.achtDelen.kolom5);
    Verwissel(m_geel.achtDelen.kolom0,
              m_geel.achtDelen.kolom6);
    Verwissel(m_rood.achtDelen.kolom2,
              m_rood.achtDelen.kolom4);
    Verwissel(m_rood.achtDelen.kolom1,
              m_rood.achtDelen.kolom5);
    Verwissel(m_rood.achtDelen.kolom0,
              m_rood.achtDelen.kolom6);
}

void Bord::Verwissel(r_n1 x,
                     r_n1 y) const
{
    c_n1 r=x;
    x=y;
    y=r;
}
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  Gebruik.h
// Implemen: Gebruik.cp
//=======================================

#pragma once

#include "Afmeting.h"

union Gebruik;
    typedef Gebruik* p_Gebruik;
    typedef const p_Gebruik cp_Gebruik;
typedef const Gebruik c_Gebruik;
    typedef c_Gebruik* pc_Gebruik;
    typedef const pc_Gebruik cpc_Gebruik;
typedef Gebruik& r_Gebruik;
typedef p_Gebruik& rp_Gebruik;
typedef cp_Gebruik& rcp_Gebruik;
typedef c_Gebruik& rc_Gebruik;
typedef pc_Gebruik& rpc_Gebruik;
typedef cpc_Gebruik& rcpc_Gebruik;

// opm: big endian.

union Gebruik
{
    n8 eenDeel;
    n1 tabel[aKolom];
    struct
        {
        n1 aVrij0;
        n1 aVrij1;
        n1 aVrij2;
        n1 aVrij3;
        n1 aVrij4;
        n1 aVrij5;
        n1 aVrij6;
        n1;
        } achtDelen;
};
```

```
//=========================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  Gebruik.cp
//=========================================

#include "Gebruik.h"
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  SpeelBord.h
// Implemen: SpeelBord.cp
//=======================================

#pragma once

#include "Bord.h"
#include "Gebruik.h"

class SpeelBord;
    typedef SpeelBord* p_SpeelBord;
        typedef p_SpeelBord* pp_SpeelBord;
        typedef const pp_SpeelBord cpp_SpeelBord;
    typedef const p_SpeelBord cp_SpeelBord;
        typedef cp_SpeelBord* pcp_SpeelBord;
        typedef const pcp_SpeelBord cpcp_SpeelBord;
typedef const SpeelBord c_SpeelBord;
    typedef c_SpeelBord* pc_SpeelBord;
        typedef pc_SpeelBord* ppc_SpeelBord;
        typedef const ppc_SpeelBord cppc_SpeelBord;
    typedef const pc_SpeelBord cpc_SpeelBord;
        typedef cpc_SpeelBord* pcpc_SpeelBord;
        typedef const pcpc_SpeelBord cpcpc_SpeelBord;
typedef SpeelBord& r_SpeelBord;
typedef p_SpeelBord& rp_SpeelBord;
typedef pp_SpeelBord& rpp_SpeelBord;
typedef cpp_SpeelBord& rcpp_SpeelBord;
typedef cp_SpeelBord& rcp_SpeelBord;
typedef pcp_SpeelBord& rpcp_SpeelBord;
typedef cpcp_SpeelBord& rcpcp_SpeelBord;
typedef c_SpeelBord& rc_SpeelBord;
typedef pc_SpeelBord& rpc_SpeelBord;
typedef ppc_SpeelBord& rppc_SpeelBord;
typedef cppc_SpeelBord& rcppc_SpeelBord;
typedef cpc_SpeelBord& rcpc_SpeelBord;
typedef pcpc_SpeelBord& rpcpc_SpeelBord;
typedef cpcpc_SpeelBord& rcpcpc_SpeelBord;

class SpeelBord : public Bord
{
private:
```

```
    // Afleiding.
    typedef Bord basis;

    // Eigen data.
    Gebruik m_aVrij;

    // Constructie / Kopie / Destructie.
public:
    SpeelBord();
    SpeelBord(rc_SpeelBord origineel);
    ~SpeelBord();

    // Verboden.
private:
    void operator=(rc_SpeelBord origineel);

    // Manipulators.
public:
    bool IsSymmetrisch() const;
    void ZoekEenMogelijkeZetSymmetrisch(r_z4 zet) const;
    void ZoekEenMogelijkeZetAsymmetrisch(r_z4 zet) const;
    void ZoekMogelijkeZettenSymmetrisch(r_z4 aZet,
                                        p_z4 zet) const;
    void ZoekMogelijkeZettenAsymmetrisch(r_z4 aZet,
                                         p_z4 zet) const;
    void DoeGeleZet(z4 zet);
    void DoeRodeZet(z4 zet);
private:
    void DoeZet(z4 zet,
                r_Raat kleur);
};
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  SpeelBord.cp
//=======================================

#include "SpeelBord.h"

#pragma profile off

SpeelBord::SpeelBord()
{
    m_aVrij.eenDeel=0x0606060606060600;
}

SpeelBord::SpeelBord(rc_SpeelBord origineel)
    : basis(origineel)
{
    m_aVrij.eenDeel=origineel.m_aVrij.eenDeel;
}

SpeelBord::~SpeelBord()
{
}

bool SpeelBord::IsSymmetrisch() const
{
    if (m_aVrij.achtDelen.aVrij2!=m_aVrij.achtDelen.aVrij4)
        {
        return false;
        }
    if (m_aVrij.achtDelen.aVrij1!=m_aVrij.achtDelen.aVrij5)
        {
        return false;
        }
    if (m_aVrij.achtDelen.aVrij0!=m_aVrij.achtDelen.aVrij6)
        {
        return false;
        }
    if (!basis::IsSymmetrisch())
        {
        return false;
        }
    return true;
```

```
}

void SpeelBord::ZoekEenMogelijkeZetSymmetrisch(r_z4 zet) const
{
    if (m_aVrij.achtDelen.aVrij3)
        {
        zet=3;
        return;
        }
    if (m_aVrij.achtDelen.aVrij2)
        {
        zet=2;
        return;
        }
    if (m_aVrij.achtDelen.aVrij1)
        {
        zet=1;
        return;
        }
    if (m_aVrij.achtDelen.aVrij0)
        {
        zet=0;
        return;
        }
}

void SpeelBord::ZoekEenMogelijkeZetAsymmetrisch(r_z4 zet) const
{
    if (m_aVrij.achtDelen.aVrij3)
        {
        zet=3;
        return;
        }
    if (m_aVrij.achtDelen.aVrij2)
        {
        zet=2;
        return;
        }
    if (m_aVrij.achtDelen.aVrij4)
        {
        zet=4;
        return;
        }
    if (m_aVrij.achtDelen.aVrij1)
        {
```

```
        zet=1;
        return;
        }
    if (m_aVrij.achtDelen.aVrij5)
        {
        zet=5;
        return;
        }
    if (m_aVrij.achtDelen.aVrij0)
        {
        zet=0;
        return;
        }
    if (m_aVrij.achtDelen.aVrij6)
        {
        zet=6;
        return;
        }
}

void SpeelBord::ZoekMogelijkeZettenSymmetrisch(r_z4 aZet,
                                               cp_z4 zet) const
{
    p_z4 p=zet;
    if (m_aVrij.achtDelen.aVrij3)
        {
        *p++=3;
        }
    if (m_aVrij.achtDelen.aVrij2)
        {
        *p++=2;
        }
    if (m_aVrij.achtDelen.aVrij1)
        {
        *p++=1;
        }
    if (m_aVrij.achtDelen.aVrij0)
        {
        *p++=0;
        }
    aZet=static_cast<z4>(p)-static_cast<z4>(zet);
    aZet>>=2;
}

void SpeelBord::ZoekMogelijkeZettenAsymmetrisch(r_z4 aZet,
```

```
                                                    cp_z4 zet) const
{
    p_z4 p=zet;
    if (m_aVrij.achtDelen.aVrij3)
        {
        *p++=3;
        }
    if (m_aVrij.achtDelen.aVrij2)
        {
        *p++=2;
        }
    if (m_aVrij.achtDelen.aVrij4)
        {
        *p++=4;
        }
    if (m_aVrij.achtDelen.aVrij1)
        {
        *p++=1;
        }
    if (m_aVrij.achtDelen.aVrij5)
        {
        *p++=5;
        }
    if (m_aVrij.achtDelen.aVrij0)
        {
        *p++=0;
        }
    if (m_aVrij.achtDelen.aVrij6)
        {
        *p++=6;
        }
    aZet=static_cast<z4>(p)-static_cast<z4>(zet);
    aZet>>=2;
}

void SpeelBord::DoeGeleZet(c_z4 zet)
{
    DoeZet(zet,
           m_geel);
}

void SpeelBord::DoeRodeZet(c_z4 zet)
{
    DoeZet(zet,
           m_rood);
```

```
}

void SpeelBord::DoeZet(c_z4 zet,
                       r_Raat kleur)
{
    r_n1 aVrij=m_aVrij.tabel[zet];
    aVrij--;
    c_n8 bit=BORDBIT(aVrij,zet);
    kleur.eenDeel or_eq bit;
}
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  SymmetrieBord.h
// Implemen: SymmetrieBord.cp
//=======================================

#pragma once

#include "SpeelBord.h"

class SymmetrieBord;
    typedef SymmetrieBord* p_SymmetrieBord;
        typedef p_SymmetrieBord* pp_SymmetrieBord;
        typedef const pp_SymmetrieBord cpp_SymmetrieBord;
    typedef const p_SymmetrieBord cp_SymmetrieBord;
        typedef cp_SymmetrieBord* pcp_SymmetrieBord;
        typedef const pcp_SymmetrieBord cpcp_SymmetrieBord;
typedef const SymmetrieBord c_SymmetrieBord;
    typedef c_SymmetrieBord* pc_SymmetrieBord;
        typedef pc_SymmetrieBord* ppc_SymmetrieBord;
        typedef const ppc_SymmetrieBord cppc_SymmetrieBord;
    typedef const pc_SymmetrieBord cpc_SymmetrieBord;
        typedef cpc_SymmetrieBord* pcpc_SymmetrieBord;
        typedef const pcpc_SymmetrieBord cpcpc_SymmetrieBord;
typedef SymmetrieBord& r_SymmetrieBord;
typedef p_SymmetrieBord& rp_SymmetrieBord;
typedef pp_SymmetrieBord& rpp_SymmetrieBord;
typedef cpp_SymmetrieBord& rcpp_SymmetrieBord;
typedef cp_SymmetrieBord& rcp_SymmetrieBord;
typedef pcp_SymmetrieBord& rpcp_SymmetrieBord;
typedef cpcp_SymmetrieBord& rcpcp_SymmetrieBord;
typedef c_SymmetrieBord& rc_SymmetrieBord;
typedef pc_SymmetrieBord& rpc_SymmetrieBord;
typedef ppc_SymmetrieBord& rppc_SymmetrieBord;
typedef cppc_SymmetrieBord& rcppc_SymmetrieBord;
typedef cpc_SymmetrieBord& rcpc_SymmetrieBord;
typedef pcpc_SymmetrieBord& rpcpc_SymmetrieBord;
typedef cpcpc_SymmetrieBord& rcpcpc_SymmetrieBord;

class SymmetrieBord : public SpeelBord
{
private:
    // Afleiding.
```

```
    typedef SpeelBord basis;

    // Globale data.
    static z4 s_aSymmetrisch;

    // Eigen data.
    bool m_isAsymmetrisch;
    z4 m_aGeelLinks;
    z4 m_aGeelRechts;
    z4 m_aRoodLinks;
    z4 m_aRoodRechts;

    // Constructie / Kopie / Destructie.
public:
    SymmetrieBord();
    SymmetrieBord(rc_SymmetrieBord origineel);
    ~SymmetrieBord();

    // Verboden.
private:
    void operator=(rc_SymmetrieBord origineel);

    // Manipulators.
public:
    bool IsSymmetrisch();
    void ZoekEenMogelijkeZet(r_z4 zet);
    void ZoekMogelijkeZetten(r_z4 aZet,
                             p_z4 zet);
    void DoeGeleZet(z4 zet);
    void DoeRodeZet(z4 zet);
};
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  SymmetrieBord.cp
//=======================================

#include "SymmetrieBord.h"

#pragma profile off

z4 SymmetrieBord::s_aSymmetrisch=0;

SymmetrieBord::SymmetrieBord()
    : m_isAsymmetrisch(false),
      m_aGeelLinks(0),
      m_aGeelRechts(0),
      m_aRoodLinks(0),
      m_aRoodRechts(0)
{
}

SymmetrieBord::SymmetrieBord(rc_SymmetrieBord origineel)
    : basis(origineel),
      m_isAsymmetrisch(origineel.m_isAsymmetrisch),
      m_aGeelLinks(origineel.m_aGeelLinks),
      m_aGeelRechts(origineel.m_aGeelRechts),
      m_aRoodLinks(origineel.m_aRoodLinks),
      m_aRoodRechts(origineel.m_aRoodRechts)
{
}

SymmetrieBord::~SymmetrieBord()
{
}

bool SymmetrieBord::IsSymmetrisch()
{
    if (m_isAsymmetrisch)
        {
        return false;
        }
    if (m_aGeelLinks!=m_aGeelRechts)
        {
        return false;
```

```
        }
    if (m_aRoodLinks!=m_aRoodRechts)
        {
        return false;
        }
    if (!basis::IsSymmetrisch())
        {
        m_isAsymmetrisch=true;
        return false;
        }
    s_aSymmetrisch++;
    return true;
}

void SymmetrieBord::ZoekEenMogelijkeZet(r_z4 zet)
{
    if (IsSymmetrisch())
        {
        ZoekEenMogelijkeZetSymmetrisch(zet);
        }
    else
        {
        ZoekEenMogelijkeZetAsymmetrisch(zet);
        }
}

void SymmetrieBord::ZoekMogelijkeZetten(r_z4 aZet,
                                        cp_z4 zet)
{
    if (IsSymmetrisch())
        {
        ZoekMogelijkeZettenSymmetrisch(aZet,
                                       zet);
        }
    else
        {
        ZoekMogelijkeZettenAsymmetrisch(aZet,
                                        zet);
        }
}

void SymmetrieBord::DoeGeleZet(c_z4 zet)
{
    if (zet<3)
        {
```

```
        m_aGeelLinks++;
        }
    else if (zet>3)
        {
        m_aGeelRechts++;
        }
    basis::DoeGeleZet(zet);
}

void SymmetrieBord::DoeRodeZet(c_z4 zet)
{
    if (zet<3)
        {
        m_aRoodLinks++;
        }
    else if (zet>3)
        {
        m_aRoodRechts++;
        }
    basis::DoeRodeZet(zet);
}
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  BordenLijst.h
// Implemen: BordenLijst.cp
//=======================================

#pragma once

#include "ZoekBord.h"

class BordenLijst;
    typedef BordenLijst* p_BordenLijst;
    typedef const p_BordenLijst cp_BordenLijst;
typedef const BordenLijst c_BordenLijst;
    typedef c_BordenLijst* pc_BordenLijst;
    typedef const pc_BordenLijst cpc_BordenLijst;
typedef BordenLijst& r_BordenLijst;
typedef p_BordenLijst& rp_BordenLijst;
typedef cp_BordenLijst& rcp_BordenLijst;
typedef c_BordenLijst& rc_BordenLijst;
typedef pc_BordenLijst& rpc_BordenLijst;
typedef cpc_BordenLijst& rcpc_BordenLijst;

class BordenLijst
{
private:
    // Eigen data.
    p_ZoekBord m_eerste;
    p_ZoekBord m_laatste;

    // Constructie / Kopie / Destructie.
public:
    BordenLijst();
    ~BordenLijst();

    // Verboden.
private:
    BordenLijst(rc_BordenLijst origineel);
    void operator=(rc_BordenLijst origineel);

    // Manipulators.
public:
    void VoegToe(p_ZoekBord bord);
```

```
    bool HaalAf(rp_ZoekBord bord);
};
```

```
//=========================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  BordenLijst.cp
//=========================================

#include "BordenLijst.h"

#pragma profile off

BordenLijst::BordenLijst()
    : m_eerste(nil),
      m_laatste(nil)
{
}

BordenLijst::~BordenLijst()
{
    while (m_eerste)
        {
        cp_ZoekBord volgende=m_eerste->m_volgende;
        delete m_eerste;
        m_eerste=volgende;
        }
}

void BordenLijst::VoegToe(cp_ZoekBord bord)
{
    bord->m_volgende=nil;
    if (m_laatste)
        {
        m_laatste->m_volgende=bord;
        }
    else
        {
        m_eerste=bord;
        }
    m_laatste=bord;
}

bool BordenLijst::HaalAf(rp_ZoekBord bord)
{
    if (!m_eerste)
        {
```

```
        return false;
        }
    bord=m_eerste;
    if (m_eerste==m_laatste)
        {
        m_eerste=nil;
        m_laatste=nil;
        }
    else
        {
        m_eerste=bord->m_volgende;
        }
    return true;
}
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  ZoekBord.h
// Implemen: ZoekBord.cp
//=======================================

#pragma once

#include "GeheugenLijst.h"
#include "GeleWinstUitvoerLijst.h"
#include "SymmetrieBord.h"

class ZoekBord;
    typedef ZoekBord* p_ZoekBord;
        typedef p_ZoekBord* pp_ZoekBord;
        typedef const pp_ZoekBord cpp_ZoekBord;
    typedef const p_ZoekBord cp_ZoekBord;
        typedef cp_ZoekBord* pcp_ZoekBord;
        typedef const pcp_ZoekBord cpcp_ZoekBord;
typedef const ZoekBord c_ZoekBord;
    typedef c_ZoekBord* pc_ZoekBord;
        typedef pc_ZoekBord* ppc_ZoekBord;
        typedef const ppc_ZoekBord cppc_ZoekBord;
    typedef const pc_ZoekBord cpc_ZoekBord;
        typedef cpc_ZoekBord* pcpc_ZoekBord;
        typedef const pcpc_ZoekBord cpcpc_ZoekBord;
typedef ZoekBord& r_ZoekBord;
typedef p_ZoekBord& rp_ZoekBord;
typedef pp_ZoekBord& rpp_ZoekBord;
typedef cpp_ZoekBord& rcpp_ZoekBord;
typedef cp_ZoekBord& rcp_ZoekBord;
typedef pcp_ZoekBord& rpcp_ZoekBord;
typedef cpcp_ZoekBord& rcpcp_ZoekBord;
typedef c_ZoekBord& rc_ZoekBord;
typedef pc_ZoekBord& rpc_ZoekBord;
typedef ppc_ZoekBord& rppc_ZoekBord;
typedef cppc_ZoekBord& rcppc_ZoekBord;
typedef cpc_ZoekBord& rcpc_ZoekBord;
typedef pcpc_ZoekBord& rpcpc_ZoekBord;
typedef cpcpc_ZoekBord& rcpcpc_ZoekBord;

class ZoekBord : public SymmetrieBord
{
```

```
private:
    // Afleiding.
    typedef SymmetrieBord basis;

    // Globale data.
    static c_z4 miljoen=1000000;
    static c_z4 maxBord=(aNiveau+1)*aKolom;
    static p_char s_borden;
    static p_ZoekBord s_vrij;
    static n4 s_tabel[256];
    static GeheugenLijst s_geheugenLijst;
    static GeleWinstUitvoerLijst s_geleWinstUitvoerLijst;
    static z4 s_aZetMiljoen;
    static z4 s_aZet;
    static z4 s_aHerinnerd;

    // Globale functies.
public:
    static void Init_klasse();
    static void Exit_klasse();
private:
    static void Init_alloc();
    static void Exit_alloc();
public:
    static void Init_voortgang();
    static void SchrijfVoortgang();
    static void SchrijfStatistiek();

    // Alloc.
    p_void operator new(n4 lengte);
    void operator delete(p_void ptr);

    // Eigen data.
private:
    cpc_ZoekBord i_vorigBord;
    z4 m_niveau;
    z4 m_laatsteIndex;
    z4 m_laatsteZet;

    // Friends.
    friend class BordenLijst;
    p_ZoekBord m_volgende;

    // Constructie / Kopie / Destructie.
public:
```

```
    ZoekBord();
    ZoekBord(rc_ZoekBord origineel);
    ~ZoekBord();

    // Verboden.
private:
    void operator=(rc_ZoekBord origineel);

    // Manipulators.
    n4 BerekenCRC() const;
    void Tel() const;
public:
    void Schrijf() const;
private:
    void SchrijfPad() const;
public:
    void DoeGeleZet(z4 index,
                    z4 zet);
    void DoeRodeZet(z4 index,
                    z4 zet);
    void ZoekGeleZet(r_Uitslag uitslag,
                     r_z4 besteZet);
    void ZoekRodeZet(r_Uitslag uitslag,
                     r_z4 besteZet);
private:
    void ProbeerAlleGeleZetten(r_Uitslag uitslag,
                               r_z4 besteZet);
    void ProbeerAlleRodeZetten(r_Uitslag uitslag,
                               r_z4 besteZet);
};
```

```
//=======================================
// Auteur:   © 2017 Cliff Huylebroeck
// Compiler: MSL
// Project:  VierOpEenRij
// Bestand:  ZoekBord.cp
//=======================================

#include "ZoekBord.h"

#include "BordenLijst.h"

#pragma profile off

p_char ZoekBord::s_borden=nil;

p_ZoekBord ZoekBord::s_vrij=nil;

n4 ZoekBord::s_tabel[256];

GeheugenLijst ZoekBord::s_geheugenLijst;

GeleWinstUitvoerLijst ZoekBord::s_geleWinstUitvoerLijst;

z4 ZoekBord::s_aZet=0;

z4 ZoekBord::s_aZetMiljoen=0;

z4 ZoekBord::s_aHerinnerd=0;

void ZoekBord::Init_klasse()
{
    Init_alloc();

    n4 crc;
    for (z4 i=0;i<256;i++)
        {
        crc=static_cast<n4>(i);
        for (z4 j=8;j>0;j--)
            {
            if (crc bitand 1)
                {
                crc=static_cast<n4>((crc>>1) xor 0xedb88320);
                }
            else
                {
```

```
                crc>>=1;
                }
            }
        s_tabel[i]=crc;
        }

    s_geheugenLijst.Init_object();

    s_geleWinstUitvoerLijst.Init_object();
}

void ZoekBord::Exit_klasse()
{
    Exit_alloc();

    s_geheugenLijst.Exit_object();

    while (s_geleWinstUitvoerLijst.IsBezig())
        {
        cout << "Wacht 1 seconde.";
        cout << endl;
        sleep(1);
        }

    s_geleWinstUitvoerLijst.Exit_object();
}

void ZoekBord::Init_alloc()
{
    c_n4 lengte=sizeof(ZoekBord);
    c_z4 totaleLengte=maxBord*lengte;
    s_borden=new char[totaleLengte];
    if (!s_borden)
        {
        Exit("Te weinig geheugen.");
        }
    s_vrij=reinterpret_cast<p_ZoekBord>(s_borden);
    p_ZoekBord p=s_vrij;
    cp_ZoekBord laatste=p+maxBord-1;
    laatste->m_volgende=nil;
    while (p<laatste)
        {
        cp_ZoekBord volgende=p+1;
        p->m_volgende=volgende;
        p=volgende;
```

```
        }
}

void ZoekBord::Exit_alloc()
{
    delete [] s_borden;
}

void ZoekBord::Init_voortgang()
{
    s_aZet=0;
    s_aZetMiljoen=0;
    s_aHerinnerd=0;
}

void ZoekBord::SchrijfVoortgang()
{
    cout << "(";
    cout << s_aZetMiljoen;
    cout << " miljoen";
    if (s_aZet)
        {
        cout << " ";
        cout << s_aZet;
        }
    cout << ";";
    cout << s_aHerinnerd;
    cout << ")";
}

void ZoekBord::SchrijfStatistiek()
{
    SchrijfVoortgang();
    cout << endl;

    s_geheugenLijst.SchrijfStatistiek();
}

p_void ZoekBord::operator new(n4)
{
    if (!s_vrij)
        {
        Exit("Te weinig geheugen.");
        }
    cp_ZoekBord gevonden=s_vrij;
```

```
    s_vrij=gevonden->m_volgende;
    gevonden->m_volgende=nil;
    return gevonden;
}

void ZoekBord::operator delete(p_void ptr)
{
    cp_ZoekBord teWissen=reinterpret_cast<p_ZoekBord>(ptr);
    teWissen->m_volgende=s_vrij;
    s_vrij=teWissen;
}

ZoekBord::ZoekBord()
    : i_vorigBord(nil),
      m_niveau(0),
      m_laatsteIndex(aKolom),
      m_laatsteZet(aKolom)
{
}

ZoekBord::ZoekBord(rc_ZoekBord origineel)
    : basis(origineel),
      i_vorigBord(&origineel),
      m_niveau(origineel.m_niveau),
      m_laatsteIndex(origineel.m_laatsteIndex),
      m_laatsteZet(origineel.m_laatsteZet)
{
}

ZoekBord::~ZoekBord()
{
}

n4 ZoekBord::BerekenCRC() const
{
    #define CALC(x) \
    static_cast<n4>(s_tabel[(crc xor x) bitand 0xff] xor (crc>>8))

    pc_n1 p;

    n4 crc=0xffffffff;

    p=reinterpret_cast<pc_n1>(&m_geel.achtDelen.kolom6);
    crc=CALC(*p++);
    crc=CALC(*p++);
```

```
    crc=CALC(*p++);
    crc=CALC(*p++);
    crc=CALC(*p++);
    crc=CALC(*p++);
    crc=CALC(*p);

    p=reinterpret_cast<pc_n1>(&m_rood.achtDelen.kolom6);
    crc=CALC(*p++);
    crc=CALC(*p++);
    crc=CALC(*p++);
    crc=CALC(*p++);
    crc=CALC(*p++);
    crc=CALC(*p++);
    crc=CALC(*p);

    return crc;
}

void ZoekBord::Tel() const
{
    s_aZet++;
    if (s_aZet<miljoen)
        {
        return;
        }

    s_aZet=0;
    s_aZetMiljoen++;

    static n1 aLijn=0;
    aLijn++;
    if (aLijn==20)
        {
        aLijn=0;
        clrscr();
        }

    SchrijfPad();
}

void ZoekBord::Schrijf() const
{
    for (z4 rij=0;rij<aRij;rij++)
        {
        for (z4 kolom=0;kolom<aKolom;kolom++)
```

```
            {
            c_n8 bit=BORDBIT(rij,kolom);
            if (m_geel.eenDeel bitand bit)
                {
                cout << "|o";
                }
            else if (m_rood.eenDeel bitand bit)
                {
                cout << "|•";
                }
            else
                {
                cout << "| ";
                }
            }
        cout << "|";
        cout << endl;
        }
    cout << "|-------------|";
    cout << endl;
}

void ZoekBord::SchrijfPad() const
{
    SchrijfVoortgang();

    // opm: hier liever geen recursie.
    pc_ZoekBord borden[aNiveau];
    pc_ZoekBord p=this;
    while (p)
        {
        cpc_ZoekBord vorigBord=p->i_vorigBord;
        borden[p->m_niveau]=p;
        p=vorigBord;
        }

    ppc_ZoekBord q=borden+1;
    for (z4 i=1;i<=m_niveau;i++,q++)
        {
        cout << " ";
        c_z1 eerste=(i bitand 1) ? '0' : 'A';
        c_n1 karakter=static_cast<n1>((*q)->m_laatsteIndex+eerste);
        cout << karakter;
        }
```

```
    cout << endl;
}

void ZoekBord::DoeGeleZet(c_z4 index,
                          c_z4 zet)
{
    m_niveau++;
    m_laatsteIndex=index;
    m_laatsteZet=zet;
    basis::DoeGeleZet(zet);
    Tel();
}

void ZoekBord::DoeRodeZet(c_z4 index,
                          c_z4 zet)
{
    m_niveau++;
    m_laatsteIndex=index;
    m_laatsteZet=zet;
    basis::DoeRodeZet(zet);
    Tel();
}

void ZoekBord::ZoekGeleZet(r_Uitslag uitslag,
                           r_z4 besteZet)
{
    #ifdef __DEBUG__
        if (s_aZetMiljoen>=3)
            {
            uitslag=roodWint;
            besteZet=7;
            return;
            }
    #endif

    c_n4 crc=BerekenCRC();

    if (s_geheugenLijst.Herinner(m_niveau,
                                 crc,
                                 m_geel.eenDeel,
                                 m_rood.eenDeel,
                                 uitslag,
                                 besteZet))
        {
        s_aHerinnerd++;
```

```
        return;
        }

    ProbeerAlleGeleZetten(uitslag,
                          besteZet);

    s_geheugenLijst.Onthoud(m_niveau,
                            crc,
                            m_geel.eenDeel,
                            m_rood.eenDeel,
                            uitslag,
                            besteZet);

    if (uitslag==geelWint)
        {
        SpeelBord bord(*this);
        bord.DoeGeleZet(besteZet);
        s_geleWinstUitvoerLijst.Schrijf(m_niveau,
                                        bord);
        }
}

void ZoekBord::ZoekRodeZet(r_Uitslag uitslag,
                           r_z4 besteZet)
{
    #ifdef __DEBUG__
        if (s_aZetMiljoen>=3)
            {
            uitslag=roodWint;
            besteZet=7;
            return;
            }
    #endif

    c_n4 crc=BerekenCRC();

    if (s_geheugenLijst.Herinner(m_niveau,
                                 crc,
                                 m_geel.eenDeel,
                                 m_rood.eenDeel,
                                 uitslag,
                                 besteZet))
        {
        s_aHerinnerd++;
        return;
```

```
        }

    ProbeerAlleRodeZetten(uitslag,
                          besteZet);

    s_geheugenLijst.Onthoud(m_niveau,
                            crc,
                            m_geel.eenDeel,
                            m_rood.eenDeel,
                            uitslag,
                            besteZet);
}

void ZoekBord::ProbeerAlleGeleZetten(r_Uitslag uitslag,
                                     r_z4 besteZet)
{
    z4 aZet;
    z4 zet[aKolom];
    ZoekMogelijkeZetten(aZet,
                        zet);

    z4 i;
    z4 zet_i;
    z4 dummy_zet;

    if (m_niveau<7)
        {
        // opm: er kan nog niet gewonnen zijn.
        for (i=0;i<aZet;i++)
            {
            zet_i=zet[i];
            ZoekBord bord(*this);
            bord.DoeGeleZet(i,
                            zet_i);
            bord.ZoekRodeZet(uitslag,
                             dummy_zet);
            if (uitslag==geelWint)
                {
                besteZet=zet_i;
                return;
                }
            }
        uitslag=roodWint;
        besteZet=*zet;
        return;
```

```
        }

    if (aZet==1)
        {
        besteZet=*zet;
        ZoekBord bord(*this);
        bord.DoeGeleZet(0,
                        besteZet);
        if (bord.GeelWint())
            {
            uitslag=geelWint;
            return;
            }
        bord.ZoekRodeZet(uitslag,
                         dummy_zet);
        return;
        }

    BordenLijst teEvalueren;
    p_ZoekBord bord;

    for (i=0;i<aZet;i++)
        {
        zet_i=zet[i];
        bord=new ZoekBord(*this);
        if (!bord)
            {
            Exit("Te weinig geheugen.");
            }
        bord->DoeGeleZet(i,
                         zet_i);
        if (bord->GeelWint())
            {
            uitslag=geelWint;
            besteZet=zet_i;
            delete bord;
            return;
            }
        teEvalueren.VoegToe(bord);
        }

    for (i=0;i<aZet;i++)
        {
        zet_i=zet[i];
        SpeelBord test(*this);
```

```
        test.DoeRodeZet(zet_i);
        Tel();
        if (test.RoodWint())
            {
            ZoekBord bord(*this);
            bord.DoeGeleZet(i,
                            zet_i);
            bord.ZoekRodeZet(uitslag,
                             dummy_zet);
            besteZet=zet_i;
            return;
            }
        }

    // opm: geel is begonnen, dus kan rood nog zetten.
    while (teEvalueren.HaalAf(bord))
        {
        bord->ZoekRodeZet(uitslag,
                          dummy_zet);
        if (uitslag==geelWint)
            {
            besteZet=bord->m_laatsteZet;
            delete bord;
            return;
            }
        delete bord;
        }

    uitslag=roodWint;
    besteZet=*zet;
}

void ZoekBord::ProbeerAlleRodeZetten(r_Uitslag uitslag,
                                     r_z4 besteZet)
{
    if (m_niveau==lNiveau)
        {
        // opm: beschouw gelijkspel als winst voor rood.
        uitslag=roodWint;
        ZoekEenMogelijkeZet(besteZet);
        return;
        }

    z4 aZet;
    z4 zet[aKolom];
```

```
    ZoekMogelijkeZetten(aZet,
                        zet);

    z4 i;
    z4 zet_i;
    z4 dummy_zet;

    if (m_niveau<8)
        {
        // opm: er kan nog niet gewonnen zijn.
        for (i=0;i<aZet;i++)
            {
            zet_i=zet[i];
            ZoekBord bord(*this);
            bord.DoeRodeZet(i,
                            zet_i);
            bord.ZoekGeleZet(uitslag,
                             dummy_zet);
            if (uitslag==roodWint)
                {
                besteZet=zet_i;
                return;
                }
            }
        uitslag=geelWint;
        besteZet=*zet;
        return;
        }

    if (aZet==1)
        {
        besteZet=*zet;
        ZoekBord bord(*this);
        bord.DoeRodeZet(0,
                        besteZet);
        if (bord.RoodWint())
            {
            uitslag=roodWint;
            return;
            }
        bord.ZoekGeleZet(uitslag,
                         dummy_zet);
        return;
        }
```

```
    BordenLijst teEvalueren;
    p_ZoekBord bord;

    for (i=0;i<aZet;i++)
        {
        zet_i=zet[i];
        bord=new ZoekBord(*this);
        if (!bord)
            {
            Exit("Te weinig geheugen.");
            }
        bord->DoeRodeZet(i,
                         zet_i);
        if (bord->RoodWint())
            {
            uitslag=roodWint;
            besteZet=zet_i;
            delete bord;
            return;
            }
        teEvalueren.VoegToe(bord);
        }

    for (i=0;i<aZet;i++)
        {
        zet_i=zet[i];
        SpeelBord test(*this);
        test.DoeGeleZet(zet_i);
        Tel();
        if (test.GeelWint())
            {
            ZoekBord bord(*this);
            bord.DoeRodeZet(i,
                            zet_i);
            bord.ZoekGeleZet(uitslag,
                             dummy_zet);
            besteZet=zet_i;
            return;
            }
        }

    // opm: m_niveau<lNiveau, dus kan geel nog zetten.
    while (teEvalueren.HaalAf(bord))
        {
        bord->ZoekGeleZet(uitslag,
```

```
                            dummy_zet);
        if (uitslag==roodWint)
            {
            besteZet=bord->m_laatsteZet;
            delete bord;
            return;
            }
        delete bord;
        }

    uitslag=geelWint;
    besteZet=*zet;
}
```

```
ZoekBord::Init_klasse();
ZoekBord bord;
Uitslag uitslag=onbekend;
z4 besteZet=255;
bord.ZoekGeleZet(uitslag,
                 besteZet);
```