4 OGP: overerving

Oefening 122

```
class Rechthoek {
   public:
      Rechthoek();
      Rechthoek(int, int);
      int omtrek() const;
      int oppervlakte() const;
      virtual void print(ostream &os) const;
   protected:
      int hoogte;
   private:
      int breedte;
};

class GekleurdeRechthoek : public Rechthoek {
   public:
      GekleurdeRechthoek();
      GekleurdeRechthoek(int, int, const string & = "onbekend");
      virtual void print(ostream &os) const;
   protected:
      string kleur;
};

class Vierkant : public Rechthoek {
   public:
      Vierkant(int = 1);
      virtual void print(ostream &os) const;
};

class GekleurdVierkant : public Vierkant , public GekleurdeRechthoek {
   public:
      GekleurdVierkant(int = 1, const string & = "onbekend");
      void print(ostream &os) const;
};

/****************** Rechthoek *********************/
Rechthoek::Rechthoek(int h, int b) : hoogte(h), breedte(b) {}
Rechthoek::Rechthoek() : Rechthoek(1,1) {}
int Rechthoek::omtrek() const {
   return (hoogte+breedte)*2;
}
int Rechthoek::oppervlakte() const { return (hoogte*breedte);}

void Rechthoek::print(ostream &os) const {
   os << "Rechthoek: " << breedte << " op " << hoogte << endl;
}

/****************** GekleurdeRechthoek *********************/
GekleurdeRechthoek::GekleurdeRechthoek(int h, int b, const string &kl)
       : Rechthoek(h,b), kleur(kl) {}
GekleurdeRechthoek::GekleurdeRechthoek() : kleur("onbekend") {}
void GekleurdeRechthoek::print(ostream &os) const {
   Rechthoek::print(os);
   os << " kleur: " << kleur << endl;
}
```

```
/****************** Vierkant *********************/
Vierkant::Vierkant(int zijde) : Rechthoek(zijde ,zijde) {}
void Vierkant::print(ostream &os) const {
   os << "Vierkant: zijde " << hoogte << endl;
}

/****************** GekleurdVierkant *********************/
GekleurdVierkant::GekleurdVierkant(int zijde , const string &kl) :
   GekleurdeRechthoek(zijde ,zijde ,kl), Vierkant(zijde) {}
void GekleurdVierkant::print(ostream &os) const {
   Vierkant::print(os);
   os << " kleur: " << kleur << endl;
}

/************Extern ****** *********************/
//moet geen friend zijn omdat print publieke methode is
ostream& operator<<(ostream &os, const Rechthoek &rh) {
   rh.print(os);
   return os;
}
//aangezien Rechthoek een "ambiguous base" is van GekleurdVierkant
//moet de operator<< specifiek herschreven worden:
ostream& operator<<(ostream &os, const GekleurdVierkant &v) {
   v.print(os);
   return os;
}
/************* aanvullingen in main voor het uitschrijven **********/
   GekleurdVierkant gv1;
   cout << gv1;
   cout << " oppervlakte: " << gv1.Vierkant::oppervlakte() << endl
        << " omtrek: " << gv1.Vierkant::omtrek() << endl;

   GekleurdVierkant gv2(12);
   gv2.print(cout);
   cout << " oppervlakte: " << gv2.Vierkant::oppervlakte() << endl
        << " omtrek: " << gv2.GekleurdeRechthoek::omtrek() << endl;

   GekleurdVierkant gv3(15,"geel");
   cout << gv3;
   cout << " oppervlakte: " << gv3.GekleurdeRechthoek::oppervlakte() << endl
        << " omtrek: " << gv3.GekleurdeRechthoek::omtrek() << endl;
```

Oefening 123

```
//aangepast hoofdprogramma:
   vector<unique_ptr<Rechthoek>> v;
   v.push_back(make_unique<Rechthoek>(4,6));
   v.push_back(make_unique<GekleurdeRechthoek>()); //gewenste type gebruiken!!!
   v.push_back(make_unique<GekleurdeRechthoek>(6,9,"rood"));
   v.push_back(make_unique<Vierkant>(10));
   for(int i=0 ; i<v.size() ; i++) {
      cout << *v[i];
      cout << " oppervlakte: " << v[i]->oppervlakte() << endl
           << " omtrek: " << v[i]->omtrek() << endl;
   }
// Alternatieve oplossing: opvullen vector met gewone pointers:
/* Nadelen:
   1) Zonder new lukt het enkel als je het toevoegen in main doet
     (en niet in procedure).
   2) Met new moet je er zelf aan denken om delete toe te voegen
*/
```

```
   Rechthoek r2(4,6);
   GekleurdeRechthoek gr1;
   GekleurdeRechthoek gr3(6,9,"rood");
   Vierkant v2(10);
   vector<Rechthoek*> v;
   v.push_back(&r2);
   v.push_back(&gr1);
   v.push_back(&gr3);
   v.push_back(&v2);
/********* OPMERKING **************/
/* met de huidige manier van overerven lukt het niet om een GekleurdVierkant
   toe te voegen, wegens dubbele overerving.
   Dit kan wel via virtuele overerving, maar dat vormt geen onderdeel van de
   leerstof. */
```

gekleurd vierkant kan wel in een pointer van gewoon vierkant worden bijgehouden

---

Oefening 124

```
#include <iostream>
#include <vector>
using namespace std;

//je kan ook containers.h inladen
template <template T>
ostream& operator<<(ostream& out, const vector<T> & v){
    out << endl << "[ ";
    for(int i=0; i<v.size()-1; i++){
        out << v[i] << " - ";
    }
    out << v[v.size()-1] << " ]" << endl;
    return out;
}

template <template T>
class mijn_vector: public vector<T>{
    using vector<T>::vector;  //constructoren gebruiken

    public:
       void verdubbel(bool herhaal_elk_element=false);
};

template<typename T>
void mijn_vector<T>::verdubbel(bool herhaal_elk_element){
     if(herhaal_elk_element){   //a b c wordt a a b b c c
          int lengte = this->size();
          //  zonder 'this->' krijg je een foutmelding!
          this->resize(2*lengte);
          for(int i=this->size()-1; i>0; i-=2){
               (*this)[i] = (*this)[i/2];
               (*this)[i-1] = (*this)[i/2];
          }
     }
     else{ //a b c wordt 2a 2b 2c
          for(int i=0; i<this->size(); i++){
               (*this)[i] = 2 * (*this)[i];
          }
     }
}
```

Oefening 125

```
void Langeslang::schrijf(ostream& out)const{
    for(int i=0; i<size(); i++){
         out<<*(operator[](i))<<" "; //of *((*this)[i])
    }
}

void Langeslang::vul(const vector<int>& v){
    resize(v.size());
    for(int i=0; i<size(); i++){
        (*this)[i] = make_unique<int>(v[i]);
    }
}

Langeslang& Langeslang::concatenate(Langeslang & c){
    int s_b = size();  //onthouden
    resize(s_b+c.size());
    if(this == &c){
        for(int i=0; i<s_b; i++){
            (*this)[i+s_b] = make_unique<int>(*c[i]);
        }
    }
    else{
        for(int i=s_b; i<size(); i++){
            (*this)[i] = move(c[i-s_b]);
        }
        c.resize(0);
    }
    return *this;
}

/***************** ALTERNATIEVE OPLOSSING ******************/
//Dit is een oplossing zonder friend-methode, en zonder gebruik te maken
//van de methode schrijf

#include "containers.h" //bevat methode om vector<T> uit te schrijven

//onderstaande methode schrijft unique_ptr<T>
//kan je ook toevoegen in containers.h

template <class T>
ostream& operator<<(ostream& out, const unique_ptr<T> &p){
     out<<*p;
     return out;
}

class Langeslang : public vector<unique_ptr<int>>{
  private:
     //      void schrijf(ostream& out) const; //niet nodig
  public:
     void vul(const vector<int>& v);
     Langeslang& concatenate(Langeslang & c);
     //niet nodig:
     /* friend ostream& operator<<(ostream& out, const Langeslang& l){
        l.schrijf(out);
        return out;
        }
     */
};
```

```
void Langeslang::vul(const vector<int>&v){
     reserve(v.size());
     for(int i = 0; i< v.size();i++){
          (*this).push_back(make_unique<int>(v[i]));
     }
}

Langeslang& Langeslang::concatenate(Langeslang & c){
     reserve(size()+c.size());
     if (this != &c){
          for(int i = 0; i< c.size();i++){
               (*this).push_back(move(c[i]));
          }
          c.clear();
     }
     else {
          int aantal = size();
          for(int i=0;i<aantal;i++){
               (*this).push_back(make_unique<int>(*c[i]));
          }
     }
     return *this;
}
```

Oefening 126

```
#include <functional>

class Groep : public vector<Persoon>{
    public:
        Persoon geef_extremum(
              function<bool(const Persoon&, const Persoon &)> func){
    // index aanpassen is zuiniger; Persoon kopieren mag zeker niet!
            int  index_beste = 0;
            for(int i=1; i<size(); i++){
                if(func(operator[](i),operator[](index_beste))){
                    index_beste = i;
                }
            }
            return operator[](index_beste);
        }
};

/****** Aanvullen in main **************/

    cout << "Eerste qua naam:     " << gr.geef_extremum(
      [](const Persoon& a, const Persoon& b){return a.naam < b.naam;})
         << endl;

    cout << "Eerste qua voornaam: " << gr.geef_extremum(
      [](const Persoon& a, const Persoon& b){
            return a.voornaam < b.voornaam;}) << endl;

    cout << "Jongste:             " << gr.geef_extremum(
      [](const Persoon& a, const Persoon& b){
            return a.leeftijd < b.leeftijd;}) << endl;

    cout << "Oudste:              " << gr.geef_extremum(
      [](const Persoon& a, const Persoon& b){
            return a.leeftijd > b.leeftijd;}) << endl;
```