4 OGP: overerving

Oefening 122

Vul de klassen `Rechthoek`, `GekleurdeRechthoek`, `Vierkant` en `GekleurdVierkant` aan. Gebruik overerving waar het kan: een `GekleurdVierkant` erft zowel van `GekleurdeRechthoek` als van `Vierkant`. Zorg dat het hoofdprogramma werkt en EXACT de gegeven output oplevert: kijk goed na, het venijn zit 'm in de details!

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

class Rechthoek {
 public:
   Rechthoek();
   Rechthoek(int, int);
   ...
   // attributen voorzien voor hoogte en breedte
   // (type int)
};

// afgeleid van Rechthoek; pas aan in hoofding
class GekleurdeRechthoek {
 public:
   ...
   // enkel extra attribuut voor kleur
};

// afgeleid van Rechthoek; pas aan in hoofding
class Vierkant {
    ...
    // geen extra attributen voorzien!
};

// afgeleid van GekleurdRechthoek en Vierkant; pas aan in hoofding
class GekleurdVierkant {
    ...
    // geen extra attributen voorzien!
};

int main () {
   Rechthoek r1;
   r1.print(cout);
   cout << "  oppervlakte: " << r1.oppervlakte() << endl
        << "  omtrek: " << r1.omtrek() << endl;

   Rechthoek r2(4,6);
   cout << r2;
   cout << "  oppervlakte: " << r2.oppervlakte() << endl
        << "  omtrek: " << r2.omtrek() << endl;

   GekleurdeRechthoek gr1;
   gr1.print(cout);
   cout << "  oppervlakte: " << gr1.oppervlakte() << endl
        << "  omtrek: " << gr1.omtrek() << endl;

   GekleurdeRechthoek gr2(5,7);
   cout << gr2;
   cout << "  oppervlakte: " << gr2.oppervlakte() << endl
        << "  omtrek: " << gr2.omtrek() << endl;
```

```
    GekleurdeRechthoek gr3(6,9,"rood");
    gr3.print(cout);
    cout << "  oppervlakte: " << gr3.oppervlakte() << endl
         << "  omtrek: " << gr3.omtrek() << endl;

    Vierkant v1;
    cout << v1;
    cout << "  oppervlakte: " << v1.oppervlakte() << endl
         << "  omtrek: " << v1.omtrek() << endl;

    Vierkant v2(10);
    v2.print(cout);
    cout << "  oppervlakte: " << v2.oppervlakte() << endl
         << "  omtrek: " << v2.omtrek() << endl;

    GekleurdVierkant gv1;
    cout << gv1;
    cout << "  oppervlakte: " << ...
         << "  omtrek: " << ...

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

    GekleurdVierkant gv3(15,"geel");
    cout << gv3;
    cout << "  oppervlakte: " << ...
         << "  omtrek: " << ...
    return 0;
}
```

gr.print(cout)!! opletten op examen=> altijd virtual void print implementeren en in de hoofdklassen ostream overloaden

De gevraagde output is:

```
Rechthoek: 1 op 1
  oppervlakte: 1
  omtrek: 4
Rechthoek: 6 op 4
  oppervlakte: 24
  omtrek: 20
Rechthoek: 1 op 1
  kleur: onbekend
  oppervlakte: 1
  omtrek: 4
Rechthoek: 7 op 5
  kleur: onbekend
  oppervlakte: 35
  omtrek: 24
Rechthoek: 9 op 6
  kleur: rood
  oppervlakte: 54
  omtrek: 30
Vierkant: zijde 1
  oppervlakte: 1
  omtrek: 4
Vierkant: zijde 10
  oppervlakte: 100
  omtrek: 40
Vierkant: zijde 1
  kleur: onbekend
  oppervlakte: 1
```

```
  omtrek: 4
Vierkant: zijde 12
  kleur: onbekend
  oppervlakte: 144
  omtrek: 48
Vierkant: zijde 15
  kleur: geel
  oppervlakte: 225
  omtrek: 60
```

Merk op: de implementatie van de klassen moet zo beknopt mogelijk zijn, dus maak optimaal gebruik van overerving en overschrijf enkel de methodes die echt nodig zijn.

Oefening 123

Ga verder met de vorige oefening (122). Vervang het hoofdprogramma door deze code:

```
int main () {

   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);

   for(int i=0 ; i<v.size() ; i++) {
       cout << v[i];
       cout << " oppervlakte: " << v[i].oppervlakte() << endl
            << " omtrek: " << v[i].omtrek() << endl;
   }

   return 0;
}
```

unique_ptr<Vierkant> gv1 =
make_unique<GekleurdVierkant>(5, "groen");
Dit werkt, maar unique_ptr<GekleurdVierkant> niet => wrs
door de double inheritance van gvierkant

De output van dit programma is niet helemaal wat we willen: er is enkel sprake van rechthoeken; de kleuren worden niet uitgeschreven en het vierkant wordt ook niet alsdusdanig herkend. Pas de code aan, zodat dit wel gebeurt.

Oefening 124

Gegeven een hoofdprogramma. Schrijf de klasse `mijn_vector` die afgeleid is van de klasse `vector`. Zorg dat de output is zoals aangegeven. Een paar verklarende noten:

1. Voor elk type `T` moet je een object van type `mijn_vector<T>` kunnen aanmaken.
2. Een object van type `mijn_vector<T>` kent dezelfde lidfuncties als de klasse `vector<T>` - uiteraard.
3. Daarbovenop zal een object van de klasse `mijn_vector<T>` zichzelf kunnen verdubbelen. Kijk goed naar de output van het programma om te beslissen hoe het verdubbelen in zijn werk gaat: `[3,4,5]` kan `[3,3,4,4,5,5]` dan wel `[6,8,10]` worden.

4. Een tip: om het i-de element van de vector aan te spreken, kan je niet gewoon `[i]` schrijven; `(*this)[i]` of `operator[](i)` kan wel.

```
int main(){
    mijn_vector<int> v{10,20,30};
    cout << v;

    v.verdubbel();
    cout<<endl<<"na verdubbelen zonder parameter: " << v;
    v.verdubbel(true);
    cout<<endl<<"na verdubbelen met param true:   " << v;

    mijn_vector<int> w(v);
    cout<<endl<<"een kopie van v: " << w;

    mijn_vector<double> u(7);
    cout<<endl<<"een vector met 7 default-elt: " << u;
    for(int i=0; i<u.size(); i++){
        u[i] = i*1.1;
    }
    cout<<endl<<"na opvullen met getallen: " << u;

    u.verdubbel();
    cout<<endl<<"na verdubbelen zonder parameter: " << u;

    return 0;
}
```

De output wordt (op een paar witlijnen na):

```
[ 10 - 20 - 30 ]
na verdubbelen zonder parameter:   [ 20 - 40 - 60 ]
na verdubbelen met param true:     [ 20 - 20 - 40 - 40 - 60 - 60 ]
een kopie van v:                   [ 20 - 20 - 40 - 40 - 60 - 60 ]
een vector met 7 default-elt:      [ 0 - 0 - 0 - 0 - 0 - 0 - 0 ]
na opvullen met getallen:          [ 0 - 1.1 - 2.2 - 3.3 - 4.4 - 5.5 - 6.6 ]
na verdubbelen zonder parameter:   [ 0 - 2.2 - 4.4 - 6.6 - 8.8 - 11 - 13.2 ]
```

Oefening 125

Gegeven onderstaande klassendeclaratie en hoofdprogramma.

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

class Langeslang : public vector<unique_ptr<int>>{
    private:
        void schrijf(ostream&)const;
    public:
        void vul(const vector<int>& v);
        Langeslang& concatenate(Langeslang & c);

    friend ostream& operator<<(ostream& out, const Langeslang& l){
        l.schrijf(out);
        return out;
    }
};
```

```
int main(){
    Langeslang a;
    Langeslang b;
    Langeslang c;
    a.vul({1,2});      // via a vind je de getallen 1 en 2
    b.vul({3,4,5});    // via b vind je de getallen 3, 4 en 5
    c.vul({6,7});      // via c vind je de getallen 6 en 7
    cout<<"a: "<<a<<endl<<"b: "<<b<<endl<<"c: "<<c<<endl<<endl;

    a.concatenate(a);
    // via a vind je nu de 4 getallen 1, 2, 1 en 2

    cout<<"na a.concatenate(a)"<<endl;
    cout<<"a: "<<a<<endl<<"b: "<<b<<endl<<"c: "<<c<<endl<<endl;

    a.concatenate(b).concatenate(c);
    // via a vind je nu de 9 getallen 1, 2, 1, 2, 3, 4, 5, 6 en 7
    // b is leeg
    // c is leeg

    cout<<"na a.concatenate(b).concatenate(c)"<<endl;
    cout<<"a: "<<a<<endl<<"b: "<<b<<endl<<"c: "<<c<<endl<<endl;
    return 0;
}
```

Implementeer de lidfuncties `schrijf`, `vul` en `concatenate` (outline). (Gelukt? Geef jezelf een pluim (of meer); implementatie van `concatenate` was een testvraag uit vorige jaargangen.)

Als extra, na het afwerken van de laatste oefening uit deze reeks: wat gebeurt er als je van de klasse `Langeslang` een template-klasse maakt? Pas aan en test uit met types `double` en `string`.

Oefening 126

Vul het onderstaande hoofdprogramma aan met lambdafuncties en implementeer de lidfunctie `geef_extremum` van de klasse `Groep`.

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

class Persoon{
    public:
       string voornaam;
       string naam;
       int leeftijd;
       Persoon(const string & v, const string & n, int l):voornaam(v),naam(n),leeftijd(l){}
};

ostream& operator<<(ostream & out, const Persoon & p){
    out<<p.naam <<" "<<p.voornaam<<" ("<<p.leeftijd<<")";
    return out;
}


class Groep : public vector<Persoon>{
    public:
        // Hier komt de declaratie van de lidfunctie 'geef_extremum',
        // die een Persoon teruggeeft.
```

```
        // Deze lidfunctie krijgt als parameter een functie mee die twee
        // objecten van de klasse Persoon vergelijkt.
};

int main(){
    Groep gr;
    gr.push_back(Persoon("Ann","Nelissen",12));
    gr.push_back(Persoon("Bert","Mertens",22));
    gr.push_back(Persoon("Celle","Lauwers",55));

    cout<<"Eerste qua naam:     " << gr.geef_extremum(.......);
    cout<<"Eerste qua voornaam: " << gr.geef_extremum(.......);
    cout<<"Jongste:             " << gr.geef_extremum(.......);
    cout<<"Oudste:              " << gr.geef_extremum(.......);
    return 0;
}
```