

5 Excepties en afsluiter

Oefening 127

```
#include <stdexcept>

class bestand_niet_lang_genoege : public invalid_argument{
public:
    bestand_niet_lang_genoege(const string & message):
        invalid_argument(message) {}
};

string regel_uit_bestand(const string & bestandsnaam, int nr){
    ifstream invoer;
    invoer.open(bestandsnaam);
    if(!invoer.is_open()){
        throw bestandsnaam+" kon niet geopend worden";
    }
    string woord,magweg;
    invoer >> woord;
    getline(invoer,magweg); // rest van de regel na 'VERHAAL'
    if(woord != "VERHAAL"){
        throw woord.c_str();
    }
    string zin;
    int teller = 0;
    while(!invoer.fail() && teller<nr){
        getline(invoer,zin);
        teller++;
    }
    if(invoer.fail()){
        throw bestand_niet_lang_genoege(
            bestandsnaam+" heeft geen "+to_string(nr)+" regels.");
    }
    return zin;
}

/***** for-lus in Main aanpassen *****/

for(int i=0; i<bestandsnamen.size(); i++){
    try{
        cout << regel_uit_bestand(bestandsnamen[i]+".txt",nrs[i])
              << endl;
    }
    catch(const char* x){
        eerste_woorden += x;
        eerste_woorden += " ";
    }
    catch(const string &s){
        bestanden_niet_gevonden += s + "\n";
    }
    catch(bestand_niet_lang_genoege bnlge){
        bestanden_niet_lang_genoege += string(bnlge.what());
    }
}
```

Oefening 128

```

template <typename T>
class Doos;

template <typename T>
class Schijf {
public:
    Schijf();
    Schijf(const Schijf<T>&);
    Schijf<T>& operator=(const Schijf<T>&);
    virtual ~Schijf<T>(); //destructor steeds virtual maken!!
    //extra
    Schijf(Schijf<T>&&); //move constructor
    Schijf<T>& operator=(Schijf<T>&&); //move operator

private:
    Doos<T> *a;
};

template <typename T>
class Doos {
public:
    Doos();
    Doos(const Doos<T>&&doos);
    Doos<T>& operator=(const Doos<T>&);
    ~Doos(); //destructor steeds virtual maken!!
    //extra
    Doos(Doos<T>&&); //move constructor
    Doos<T>& operator=(Doos<T>&&); //move operator
private:
    vector<T> b;
    Schijf<T> **d;
    Doos<T> *c;
};

***** SCHIJF *****
//extra: move constructor
template<typename T>
Schijf<T> :: Schijf(Schijf<T>&& schijf) : a(schijf.a) {
    schijf.a = nullptr;
}

//extra: move operator
template <typename T>
Schijf<T>& Schijf<T> :: operator=(Schijf<T>&& schijf){
    if (this != &schijf) {
        delete a;
        a = schijf.a;
        schijf.a = nullptr;
    }
    return *this;
}

***** DOOS *****
//extra: move constructor
template<typename T>
Doos<T> :: Doos(Doos<T>&& doos) :
    b(move(doos.b)), c(doos.c), d(doos.d) {
    doos.c = nullptr;
    doos.d = nullptr;
}

```

```
}

//extra: move operator
template<typename T>
Doos<T>& Doos<T> :: operator=(Doos<T>&& doos){
    if (this!=&doos) {
        b = move(doos.b);
        c = doos.c;
        doos.c = nullptr;
        for(int i=0;i<3;i++) delete(d[i]);
        delete[] d;
        d = doos.d;
        doos.d = nullptr;
    }
    return *this;
}
```

Oefening 129

```
#include "figuren.h"
#include <memory>

class Blokkendoos : vector<unique_ptr<Figuur>>{
private:
    unique_ptr<Figuur> max_opp;
    void schrijf(ostream&) const;
public:
    Blokkendoos();
    Blokkendoos(const string & bestandsnaam);
    unique_ptr<Figuur> geef_figuur_met_grootste_oppervlakte();
    void push_back(unique_ptr<Figuur> & figuur);
    friend ostream& operator<<(ostream& out, const Blokkendoos& l){
        l.schrijf(out);
        return out;
    }
};

Blokkendoos::Blokkendoos() = default;

void Blokkendoos::push_back(unique_ptr<Figuur> & figuur){
    if(max_opp==nullptr){
        max_opp = move(figuur);
    }
    else{
        vector<unique_ptr<Figuur>>::push_back(move(figuur));
        if(max_opp->oppervlakte() < operator[](size()-1)->oppervlakte()){
            max_opp.swap(operator[](size()-1));
        }
    }
}

Blokkendoos::Blokkendoos(const string & bestandsnaam){
    unique_ptr<Figuur> up;
    ifstream input(bestandsnaam);
    string soort;
    input >> soort;
    while (!input.fail() ){
        if (soort == "rechthoek"){
            double lengte, breedte;
            input >> lengte >> breedte;
            up = make_unique<Rechthoek>(lengte,breedte);
```

```

    }
    else if(soort == "vierkant"){
        double zijde;
        input >> zijde;
        up = make_unique<Vierkant>(zijde);
    }
    else { //soort == "cirkel"
        double straal;
        input >> straal;
        up = make_unique<Cirkel>(straal);
    }
    push_back(up);
    input >> soort;
}
input.close();
}

void Blokkendoos::schrijf(ostream& out) const{
    for(int i=0; i<size(); i++){
        out<<endl<<"  "<<i<<" ";
        out<<*(operator[](i));
    }
    out<<endl<<"MAX "<<*max_opp<<endl;
    // de for-lus kan ook met for-each als je de teller toch niet vermeldt:
    /*
        for(const auto & ptr : *this){
            out<<*ptr<<endl;
        }
        */
}

unique_ptr<Figuur> Blokkendoos::geef_figuur_met_grootste_oppervlakte(){
    int index_tweedegrootste = 0;
    for(int i=1; i<size(); i++){
        if(operator[](i)->oppervlakte() >
            operator[](index_tweedegrootste)->oppervlakte()){
            index_tweedegrootste = i;
        }
    }

    operator[](index_tweedegrootste).swap(operator[](size()-1));
    // nu staat tweede grootste achteraan; die moet naar max_opp verhuizen
    unique_ptr<Figuur> hulpptr;
    hulpptr.swap(max_opp);
    //max_opp.swap(operator[](size()-1)); of gebruik (*this)[i]-notatie:
    max_opp.swap((*this)[size()-1]);
    resize(size()-1);
    return move(hulpptr);
}

int main() {
    Blokkendoos blokkendoos("figuren.txt");
    cout<<endl<<"ALLE FIGUREN: "<<blokkendoos<<endl;

    cout<<endl<<"DE 3 GROOTSTE, van groot naar klein: "<<endl;
    for(int i=0; i<3; i++){
        cout<<"figuur met grootste opp:
            "<<*blokkendoos.geef_figuur_met_grootste_oppervlakte()<<endl;
    }

    cout<<endl<<"DE NIEUWE BLOKKENDOOS BEVAT ALLEEN NOG DE KLEINERE FIGUREN: ";
    cout<<blokkendoos<<endl;
    return 0;
}

```

}