UNIVERSITEIT GENT

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

LABO Programmeren in C en C++

Oefeningenbundel

PARTIM C++

OPLOSSINGEN

Leen Brouns

Helga Naessens

Wim Van den Breen

Opleiding Industrieel Ingenieur Informatica / Elektronica / Automatisering

september 2017

REEKS A

Kennismaking met C++
cin / cout en aanverwante operatoren, (standaard)strings, templates, default parameters

Oefening 101

```
#include <iostream>
using std::cout;
using std::endl;

int main(){
    cout << "Hello world!\n";   // of cout << "Hello world!" << endl;
    for(int i=10; i>0; i--){
        cout << i << " ";
    }
    cout << endl << "START";
    return 0;
}
```

Oefening 102

```
#include <iostream>
#include <iomanip>
using std::cout;
using std::endl;
using std::setw;
using std::oct;
using std::hex;
using std::dec;
// of kortweg
// using namespace std;

int main(){
    for(int i=0; i<=64; i++){
        cout << setw(6) << oct << i << setw(6) << dec << i << setw(6) << hex << i << endl;
    }

    return 0;
}
```

Oefening 103

```
#include <iostream>
#include <string>
using std::string;
using std::cin;
using std::cout;
using std::endl;

int main(){
    const int AANT=3;  // om te testen zet je dit op een doenbaar aantal!
    int getallen[AANT];
    char letters[AANT];
    string woorden[AANT];
    cout<<"Geef "<<AANT<<" getallen, "
```

```
        <<"letters resp. woorden: ";
    // cin>>getallen;   // Een array inlezen, doe je element per element.
    cin>>letters;
    // cin>>woorden;    // Een array inlezen, doe je element per element;
    cout<<getallen<<endl; // Hier vraag je dus om het adres vd array uit te schrijven
    cout<<letters<<endl;
    cout<<woorden<<endl;  // Hier vraag je dus om het adres vd array uit te schrijven
    return 0;
}

// ENIGE UIZONDERING op de twee regels
//   "een array inlezen, doe je element per element"
//   "een array uitschrijven, doe je element per element"
// is een array van char's. Daarvoor werd de inlees-operator
// en uitschrijf-operator overschreven met de gekende functionaliteit.
```

Oefening 104

```
/*
OPLOSSING

Het eerste stukje werkt niet. De reden: de code
  "" + c
start met een constante c-string.
(Je zou kunnen argumenteren dat dit ook de notatie is voor
een constante standard-string, maar hoe moet de compiler het verschil
weten? De c-strings waren er eerst, dus de compiler interpreteert
"" als een c-string.)

Daar probeer je met de
+-operator iets bij te tellen. Maar c-strings tel je niet op
met '+', wel met strcat.

Het tweede stukje werkt wel. De reden: de code
  string w = "";
zorgt ervoor dat de constante "" (=ledige) c-string
gecast wordt naar een (standard-)string.
Strings kan je wel aan elkaar plakken met de +-operator.
*/



#include <iostream>
using namespace std;

int main(){

    char c = 'x';
    string s = "" + c;
    cout << "karakter "<< c <<" omgezet: "<< s << "." << endl;

    char k = 'y';
    string w = "";
    w += k;
    cout << "karakter "<< k <<" omgezet: "<< w << "." << endl;
    return 0;
}
```

Oefening 105

```
#include <string>
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

string genereer_string(int n){
    string s = "";
    for(int i=0; i<n; i++){
        s += ('a'+rand()%26);
    }
    return s;
}

void vul_array_met_strings(string * tab, int n, int len){
    for(int i=0; i<n; i++){
        tab[i] = genereer_string(len);
    }
}

void schrijf(const string * tab, int n){
    for(int i=0; i<n; i++){
        cout<<tab[i]<<"  -  ";
    }
}

void bepaal_min_en_max(const string * tab, int n, string & min, string & max){
    min = tab[0];
    max = tab[0];
    for(int i=1; i<n; i++){
        if (min > tab[i]){
            min = tab[i];
        }
        else if (max < tab[i]){
            max = tab[i];
        }
    }
}

int main(){

    srand(time(NULL));

    cout<<genereer_string(10)<<endl;
    cout<<genereer_string(10)<<endl;

    string tab[10];
    vul_array_met_strings(tab,10,3);
    schrijf(tab,10);

    string min, max;
    bepaal_min_en_max(tab,10,min,max);
    cout<<endl<<"min is "<<min;
    cout<<endl<<"max is "<<max<<endl;

    return 0;
}
```

Oefening 106

```
#include <iostream>
#include <string>
using std::cout;
using std::endl;
using std::string;

///////// STRUCT EN INITIALISATIE / PRINT VAN STRUCT

struct Persoon{
    string naam;
    int leeftijd;
    double lengte;
};

void initialiseer(Persoon & p, const string & naam, int leeftijd, double lengte){
    p.naam = naam;
    p.leeftijd = leeftijd;
    p.lengte = lengte;
}

void print(const Persoon& p){
    cout << p.naam << " (" << p.leeftijd << " jaar, " << (int)p.lengte <<"m" <<
        ((int)(p.lengte*100)%100) <<")";
}

////////// FUNCTIE 'grootte' VOOR VERSCHILLENDE TYPES

double grootte(double x){
    return x;
}

int grootte(const string & woord){
    return woord.size();
}

int grootte(const Persoon & p){
    return p.leeftijd;
}

/* // alternatief: grootte van persoon wordt bepaald door zijn/haar lengte
double grootte(const Persoon & p){
    return p.lengte;
}*/
/* // alternatief: grootte van persoon wordt bepaald door lengte van zijn/haar naam
int grootte(const Persoon & p){
    return grootte(p.naam);
}*/

////////// FUNCTIE 'grootste' VOOR VERSCHILLENDE TYPES VAN ARRAYS

template <class T>
T grootste(const T * array, int lengte){
    T gr = array[0];
    for(int i=1; i<lengte; i++){
        if(grootte(gr) < grootte(array[i])){
            gr = array[i];
        }
    }
    return gr;
```

```
}


//////// MAIN

int main(){
    double getallen[5] = {5.5,7.7,2.2,9.9,9.8};
    string woorden[3] = {"geloof","hoop","de liefde"};
    cout << grootste(getallen,5) << endl;
    cout << "De grootste van de drie is " << grootste(woorden,3) << "." << endl;

    Persoon personen[3];
    initialiseer(personen[0],"samuel",12,1.52);
    initialiseer(personen[1],"jente",22,1.81);
    initialiseer(personen[2],"idris",42,1.73);
    const Persoon & gr = grootste(personen,3);
    print(gr);

    return 0;
}
```

Oefening 107

```
#include <iostream>
#include <cmath>
using std::cout;
using std::endl;

/*
void schrijf(const int * array, int aantal = 1, bool achterstevoren = false){
    int start = 0;
    int stop = aantal-1;
    int stap = 1;
    int aantal_elt = 0;
    if(achterstevoren){
        aantal_elt = aantal-1;
    }
    for(int i=start; i<=stop; i+=stap){
        cout << array[abs(aantal_elt-i)] << " ";
    }
    cout << endl;
}
*/

void schrijf(const int * array, int aantal, bool achterstevoren = false, char tussenteken
    = ' '){
    if(achterstevoren){
        cout << array[aantal-1];
        for(int i=aantal-2; i>=0; i--){
            cout << tussenteken << array[i];
        }
    }
    else{
        cout << array[0];
        for(int i=1; i<aantal; i++){
            cout << tussenteken << array[i];
        }
    }
    cout << endl;
}
```

```
int main(){
    int getallen[] = {1,3,5,7,9,11,13};

    schrijf(getallen,7);
    schrijf(getallen,7,true);
    schrijf(getallen,7,false,'-');
    schrijf(getallen,7,true,'-');

    return 0;
}
```

REEKS B

Kennismaking met unique pointers, λambdafuncties en bestanden

Oefening 108

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

// GEGEVEN
void schrijf(const string * s, int aantal){
    cout<<endl;
    for(int i=0; i<aantal-1; i++){
        cout<<s[i]<<" - ";
    }
    cout<<s[aantal-1];
}

// GEGEVEN
void verwijder(string * s, int aantal, int volgnr){
    if(volgnr < aantal){
        for(int i = volgnr; i < aantal-1; i++){
            s[i] = s[i+1];
        }
    }
}


// OPGELET!!
// WAT WE NIET WILLEN ZIEN STAAN IN DE PROCEDURE HIERONDER:
//    *s[i] = *s[i+1]    --> dan kopieer je strings
// OF
//    *s[i] = move(*s[i+1])  --> dan swap je strings; probeer maar eens uit:
//                               als je 'Rein' weghaalt uit 'Rein Ada Eppo'
//                               eindig je met 'Ada Eppo Rein'

void verwijder(unique_ptr<string> * s, int aantal, int volgnr){
    if(volgnr < aantal-1){
        for(int i = volgnr; i < aantal-1; i++){
            s[i] = move(s[i+1]);
        }
    }
    else if(volgnr == aantal-1){
        s[volgnr].reset();
    }
}

void schrijf(const unique_ptr<string> * s, int aantal){
    cout<<endl;
    for(int i=0; i<aantal-1; i++){
        if(s[i]==nullptr){
            cout<<"NULL"<<" - ";
        }
        else{
            cout<<*s[i]<<" - ";
        }
    }
    if(s[aantal-1] == nullptr){
        cout<<"NULL";
```

```
    }
    else{
        cout<<*s[aantal-1];
    }
}

int main(){

    string namen[]={"Rein","Ada","Eppo"};

    schrijf(namen,3);
    verwijder(namen,3,0);
    schrijf(namen,3);


    unique_ptr<string>
       pnamen[]={make_unique<string>("Rein"),make_unique<string>("Ada"),make_unique<string>("Eppo")}

    schrijf(pnamen,3);
    verwijder(pnamen,3,0);
    schrijf(pnamen,3);

    return 0;
}
```

Oefening 109

```
#include <iostream>
#include <functional>
#include <iomanip> // voor setw
using namespace std;

void schrijf(const string & tekst, const int * v, int aantal){
    cout<<tekst;
    for(int i=0; i<aantal; i++){
        cout<<setw(4)<<v[i]<<" ";
    }
    cout<<endl;
}

void vul_array(const int * a, const int * b, int * c, int grootte, function<int(int,int)>
   func ){
    int i;
    for(i=0; i<grootte; i++){
        c[i] = func(a[i],b[i]);
    }
}


int main(){
    const int GROOTTE = 10;
    int a[] = {0,1,2,3,4,5,6,7,8,9};
    int b[] = {0,10,20,30,40,50,60,70,80,90};
    int c[GROOTTE];

    vul_array(a,b,c,GROOTTE,[](int x,int y){return x+y;});
    schrijf("SOM:      ",c,GROOTTE);
    vul_array(a,b,c,GROOTTE,[](int x,int y){return x*y;});
    schrijf("PRODUCT:  ",c,GROOTTE);
    vul_array(a,b,c,GROOTTE,[](int x,int y){return x-y;});
```

```
    schrijf("VERSCHIL: ",c,GROOTTE);

    return 0;
}
```

Oefening 110

```
#include <iostream>
#include <fstream>
using std::ifstream;
using std::cout;
using std::endl;

int main(){

    int frequentie[26] = {0};

    ifstream invoer("lord.txt");
    if(!invoer.is_open()){
        cout<<"bestand niet gevonden"<<endl;  // wordt later een exceptie
    }
    else{

        char letter;

        invoer >> letter;
        while( ! invoer.fail() ){
            if('a' <= letter && letter <= 'z'){
                frequentie[(letter-'a')] ++;
            }
            invoer >> letter;
        }

        for(int i=0; i<26; i++){
            cout<<(char)('a'+i)<<"  "<<frequentie[i]<<endl;
        }

    }
    invoer.close();

    return 0;
}
```

Oefening 111

```
#include <iostream>
#include <fstream>
#include <string>
using std::ifstream;
using std::ofstream;
using std::cout;
using std::endl;
using std::string;



int main(){
    ifstream in_1("stationnetje.txt");
    ifstream in_2("paddestoel.txt");
```

```
    ofstream uit("mix.txt");

    if(!in_1.is_open() || !in_2.is_open()){
        cout<<"minstens een bestand werd niet gevonden - jammer.... "<<endl;
    }
    else{
        int teller = 0;
        string zin_1,zin_2;
        getline(in_1,zin_1);
        getline(in_2,zin_2);
        while(!in_1.fail() && !in_2.fail()){
            if(teller%2==0){
                uit << zin_1 << endl;
            }
            else{
                uit << zin_2 << endl;
            }
            teller++;
            getline(in_1,zin_1);
            getline(in_2,zin_2);
        }
        in_1.close();
        in_2.close();
        uit.close();
    }
    return 0;
}

// De oplossing van de uitbreiding laten we aan eigen inventiviteit over.
// Allicht heb je ondertussen in de theorieles het gebruik van de container
// 'vector' gezien. Die kan wel van pas komen.
// In ieder geval: bewaar de invoerstreams in een array of een vector.
```

REEKS C

Containers

Oefening 112

```
struct Persoon{
    string naam;
    int leeftijd;
};

// gegeven:
ostream& operator<<(ostream& out, const Persoon & p){
    out<<p.naam<<" ("<<p.leeftijd<<" j)";
    return out;
}

// de schrijf-procedure voor vectoren wordt vervangen door
// de uitschrijf-operator voor vectoren:
template<typename T>
ostream& operator<<(ostream& out, const vector<T> & v){
    if(v.size()==0){
        out<<"[]";
    }
    else{
        out<<"[ ";
        for(int i=0; i<v.size()-1; i++){
            out<<v[i]<<" | ";
        }
        out<<v[v.size()-1]<<" ]";
    }
    return out;
}
```

Oefening 113

```
// Set uitschrijven? als je per se komma's tussen de elementen wil
// (en geen komma meer na het laatste element), kan je niet met een
// gewone while-lus werken. De reden: je kan niet 'rekenen' met iteratoren,
// dus " s.end() - 1 " heeft geen betekenis (al had je gehoopt dat dat
// de iterator zou zijn die op het laatste element staat te wijzen...)

// Wat je wel zou kunnen doen: een iterator 'hulp' gelijkstellen aan s.end(),
// die iterator 'hulp' dan eentje naar voor schuiven, en in de while-lus
// voortdoen zolang de iterator die vooraan startte, niet gelijk is aan
// die iterator 'hulp'.

template<typename T>
ostream& operator<<(ostream& out, const set<T> & s){
    out<<"{ ";
    typename set<T>::const_iterator it = s.begin();
    for(int i=0; i<s.size()-1; i++){
        out<<*it<<" , ";
        it++;
    }
    out<<*it<<" }";
    return out;
```

```
}

template<typename T>
ostream& operator<<(ostream& out, stack<T> st){   // st MOET kopie zijn
    while(!st.empty()){
        out<<"  "<<st.top()<<endl;
        st.pop();
    }
    return out;
}

template<typename S, typename D>
ostream& operator<<(ostream& out, const map<S,D> & m){
    typename map<S,D>::const_iterator it = m.begin();
    while(it!=m.end()){
        out<<"  "<<it->first<<" --> "<<it->second<<endl;
        it++;
    }
    return out;
}
```

Oefening 114

```
#include "opl_18_containers.h"

const int AANTAL = 5;

void oef01(){
    stack<string> st;
    st.push("een");
    st.push("twee");
    st.push("drie");
    cout<<st; // als je geen kopie had gemaakt bij uitschrijven van de stack, zal dit
    cout<<st; // hier niet werken!!! (dan heb je de 2e maal een lege stack)
}

void oef02(){
    vector<string> tabel[AANTAL];
    tabel[1].push_back("aap");
    tabel[1].push_back("noot");
    tabel[1].push_back("mies");
    for(int i=0; i<AANTAL; i++){
        cout<<tabel[i];
    }
}

void oef03(){
    vector<vector<int> > v;
    for(int i=0; i<AANTAL; i++){
        vector<int> w(i);
        for(int k=0; k<v.size(); k++){
            w[k]=10+10*k;
        }
        v.push_back(w);
    }
    //cout<<v;
    for(int i=v.size()-1; i>=0; i--){
        for(int k=v[i].size()-1; k>=0; k--){
            cout<<v[i][k]<<" ";
        }
```

```
        cout<<endl;
    }
}

int main(){
    cout<<endl<<endl<<"OEF 01"<<endl; oef01();
    cout<<endl<<endl<<"OEF 02"<<endl; oef02();
    cout<<endl<<endl<<"OEF 03"<<endl; oef03();
    return 0;
}
```

Oefening 115

```
#include "opl_18_containers.h"

int main(){
    map<char,unordered_set<string> > m;
    string woord;
    cout<<"geef woorden, eindig met STOP"<<endl;
    cin>>woord;
    while(woord!="STOP"){
        m[woord[0]].insert(woord);
        cin>>woord;
    }
    cout<<"geef een letter, ik zeg hoeveel verschillende woorden van daarnet met die
       letter starten ";
    char letter;
    cin>>letter;
    if(m.count(letter)==1){
        cout<<"er waren "<<m[letter].size()<<" verschillende woorden met die
           startletter"<<endl;
    }
    else{
        cout<<"er was geen enkel woord met die startletter"<<endl;
    }
}
```

Oefening 116

```
#include "opl_18_containers.h"

int main(){
    vector<map<char,set<string> > > v;
    string woord;
    cout<<"geef woorden, eindig met STOP"<<endl;
    cin>>woord;
    while(woord!="STOP"){
        if(woord.size()+1 > v.size()){  // hier niet vergelijken met v.size()-1
            v.resize(woord.size()+1);   // want size_t kan niet negatief worden
        }
        v[woord.size()][woord[0]].insert(woord);
        cin>>woord;
    }
    //cout<<v;

    //cout<<endl<<"geef een woord, ik zoek even lange woorden die met dezelfde letter
       starten ";
    //cin>>woord;
    woord = "sinterklaas";
```

```
    if(woord.size() < v.size()){
        if(v[woord.size()].count(woord[0])==1){
            cout<<v[woord.size()][woord[0]];
        }
        else{
            cout<<"geen woorden van die lengte met startletter "<<woord[0]<<endl;
        }
    }
    else{
        cout<<"geen woorden gevonden die zo lang zijn"<<endl;
    }
    return 0;
}
```

Oefening 117

```
/*
(a) unordered_set<string>
(b) geen container nodig; meteen uitschrijven
(c) stack<string> of vector<string>
(d) geen container nodig; als je het woord op voorhand kent: gewoon tellen
(e) idem; meteen vindplaatsen uitschrijven
(f) map<string,int>
(g) map<string,vector<int> >
*/
```

Oefening 118

```
/*
Waaraan kunnen wij zien of je een verkapte javaprogrammeur dan wel een echte
    c++-programmeur bent?
Als je ergens "="-tekens hebt staan, gebruik je de toekenningsoperator.
En die TOEKENNINGSOPERATOR die MAAKT KOPIES in C++.
En kopies maken is DUUUUUUUR.
Dus dat doe je niet zonder geldige reden.
*/

#include "opl_18_containers.h"



void
    vul_in_zoveelste_map_beeld_van_sleutel_aan_met_set_van_drie(vector<map<string,stack<set<string>>>
    & vect,int index,string sleutel,string eerste,string tweede,string derde){

    // OPDRACHT 1
    // een nieuwe set op stack die bij "noot" hoort steken:
    set<string> s;
    s.insert(eerste);
    s.insert(tweede);
    s.insert(derde);
    vect[0][sleutel].push(move(s));


    // IN C++11 OOK MOGELIJK OP 2 REGELS:
    set<string> st = {eerste,tweede,derde};
    vect[0][sleutel].push(move(st)); // liet je de 'move' weg?
                        // dan maak je nog een kopie van de set s
                        // (het gebruik van move (=std::move)
```

```
                        // komt ook nog glater in theorie aan bod)

    // ... EN ZELFS OP 1 REGEL,
    // waarbij je automatisch (zoner move) vermijdt om de set te kopieren:
    vect[0][sleutel].push({eerste,tweede,derde});

}

// wat je zeker NIET mag doen in vorige opdracht:
// een kopie nemen van v[0], bvb   map<string,stack<set<string>>> hulpmap   = vect[0];
// een kopie nemen van v[0][sleutel], bvb     stack<set<string>>  hulpstack =
    hulpmap[sleutel]
// etcetera




bool zoveelste_map_beeldt_woord_af_op_stack_waarvan_bovenste_dit_element_bevat(const
    vector<map<string,stack<set<string>>>> & v,int index, const string & woord, const
    string & element){
    bool aanwezig = false;
    map<string,stack<set<string> > >::const_iterator it = v[index].find(woord);
    if(it != v[index].end()){
        if(!it->second.empty()){
            set<string>::const_iterator it2 = it->second.top().find(element);
            if(it2 != it->second.top().end()){
                aanwezig = true;
                // woord is aanwezig, en top van bijhorende stack bevat element
            }
            else{
                aanwezig = false;
                // woord is aanwezig, maar top van bijhorende stack bevat element niet
            }
        }
        else{
            aanwezig = false;
            // woord is aanwezig, maar bijhorende stack is leeg
        }
    }
    else{
        aanwezig = false;
        // woord is niet aanwezig
    }
    return aanwezig;
}


int main(){

    vector<map<string,stack<set<string> > > > v(5);

    vul_in_zoveelste_map_beeld_van_sleutel_aan_met_set_van_drie(v,0,"noot","do","re","mi");
    cout<<v;

    if(zoveelste_map_beeldt_woord_af_op_stack_waarvan_bovenste_dit_element_bevat(v,0,"noot","re")){
        cout<<"\nmap op index 0 bevat sleutel 'noot', en element 're' zit in zijn
            bovenste set van de bijhorende stack";
    }
    else{
        cout<<"\nFOUT 1....";
    }
```

```
    if(zoveelste_map_beeldt_woord_af_op_stack_waarvan_bovenste_dit_element_bevat(v,0,"noot","sol")){
        cout<<"\nFOUT 2....";
    }
    if(zoveelste_map_beeldt_woord_af_op_stack_waarvan_bovenste_dit_element_bevat(v,0,"appelmoes","re"
        cout<<"\nFOUT 3....";
    }
    if(zoveelste_map_beeldt_woord_af_op_stack_waarvan_bovenste_dit_element_bevat(v,1,"noot","re")){
        cout<<"\nFOUT 4....";
    }

    cout<<v;



    return 0;
}
```

Oefening 119

```
#include <iostream>
#include <set>
#include <string>
#include "opl_15_containers.h"
using std::cout;
using std::string;
using std::endl;
using std::set;
using std::string;



int main(){
    string tabel[] = {"a","b","c","d","e","f","g","h","i","j","k","l","m",
                      "n","o","p","q","r","s","t","u","v","w","x","y","z"};
    int aantal = sizeof(tabel)/sizeof(string);

    set<string> verzameling;
    for(int i=0; i < aantal; i++){
        verzameling.insert(tabel[i]);
    }

    cout<<verzameling<<endl;

    set<string>::iterator it = verzameling.begin();
    for(int i=0; i < aantal; i+=3){
        cout<<i<<" ";
        /* DIT WERKT, MAAR IS INEFFICIENT (omdat 'erase' weer op zoek moet gaan)
        string xx = *it;
        it++;
        verzameling.erase(xx);
        it++;
        it++;
        */
        set<string>::iterator it_hulp = it;
        it++;
        verzameling.erase(it_hulp); // (it-1) onmogelijk
        it++;
        it++;
    }
```

```
    cout<<endl<<verzameling;

    return 0;
}

// ZAKEN DIE NIET MOGELIJK / NIET JUIST ZIJN:
//
//     it = it+3;      (kan wel voor pointers, niet voor iteratoren)
//
//     erase(it) oproepen en dan hopen dat 'it' nog een zinvolle waarde heeft
//
//     testen op 'it != verzameling.end()' als je iterator 'it' telkens
//     per 3 laat opschuiven; dan zal it in 2 van de 3 gevallen al VOORBIJ
//     'verzameling.end()' staan wijzen.
//
//     iteratoren vergelijken met < of <=
//
//     in C++ aan set::erase(it) vragen om na het verwijderen een iterator terug
//     te geven die net na het verwijderde element staat te wijzen.
//     Dit kan in C++ WEL bij de container 'list';
//     dit kan in C++11 OOK bij de container 'set'.
//
//     een for-lus gebruiken waarvan de eind-grens tussendoor wijzigt.
//     for(int i = 0; i < verzameling.size(); i++){
//         verzameling.erase(...);    --> WIJZIGT DE BOVENGRENS VAN DE LUS
//         //...
//     }
```

Oefening 120

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


void regelnummers_opslaan(map<int,string> & m, vector<int> & v, const string & bestand){
    ifstream input;
    input.open(bestand);
    if(!input.is_open()){
        cout << "bestand niet gevonden.... "<<endl;
    }
    else{
        int nr;
        input>>nr;
        while(!input.fail()){
            m[nr]="";  // manier om nr in domein te steken
            v.push_back(nr);
            input>>nr;
        }
        input.close();
    }
}
// alternatief: de getallen nog niet in het domein van de map steken, maar in een aparte
    set.
```

```
void tekstregels_opzoeken(map<int,string> & m, const string & bestand){
    ifstream input;
    input.open(bestand);
    if(!input.is_open()){
        cout << "bestand niet gevonden.... "<<endl;
    }
    else{
        string lijn;
        int nr=1;
        getline(input,lijn);
        while(!input.fail()){
            if(m.count(nr)==1){
                m[nr]=lijn;
            }
            nr++;
            getline(input,lijn);
        }
        input.close();
    }
}
// Merk op: het kan efficienter. In bovenstaande code ga je nl. voor ELKE regel
// van de bijbel navragen of deze regel bewaard moet worden.
// En NA 29454 (de laatste die je nodig hebt) lees je ook nog alles in.
// Vervang dit dus door onderstaand alternatief:
//     voor elk regelnummer dat je moet bewaren  (FOR-lus!)
//             lees eerst alles in wat er voor komt (en doe er niets mee)
//             bewaar enkel de bewuste regel zelf



// OPGELET! als je de code schrijft zoals hieronder, dan kan je niet de gewenste
// 'veiligheid' inbouwen zodat de map niet kan veranderd worden:
// hier kan geen const bij map staan, want de aanroep 'm[...]' zou eventueel iets kunnen
// wijzigen aan de map !! (zie compilermeldingen als je toch const toevoegt)
/*
void regelnummers_vervangen(map<int,string> & m, const vector<int> & v, const string &
    bestandsnaam){
    ofstream output(bestandsnaam.c_str());
    for(int i=0;i<v.size();i++){
        output<<endl<<m[v[i]];
    }
    output.close();
}
*/

// een properder oplossing voor dit probleem: zoek waar de regelnummer zich in de map
    bevindt
// (gebruik de methode 'find', dit geeft een iterator terug),
// en gebruik die iterator om de bijhorende tekst uit te schrijven.
void regelnummers_vervangen(const map<int,string> & m, const vector<int> & v, const
    string & bestandsnaam){
    ofstream output(bestandsnaam.c_str());
    for(int i=0;i<v.size();i++){
        map<int,string>::const_iterator it=m.find(v[i]);
        output<<endl<<it->second;
    }
    output.close();
}


int main(){
```

```
    //regelnummers.txt bevat {18876,10000,27132,517,8999,29454,22002,2008,27312,25712}

    vector<int> volgorde_regelnummers;
    map<int,string> tekst_op_regelnr;
    regelnummers_opslaan(tekst_op_regelnr,volgorde_regelnummers,"regelnummers.txt");
    tekstregels_opzoeken(tekst_op_regelnr,"nbible.txt");
    regelnummers_vervangen(tekst_op_regelnr,volgorde_regelnummers,"verhaal.txt");
    cout<<"Het resultaat is te zien in het bestand 'verhaal.txt'."<<endl;

    return 0;
}
```

Oefening 121

```
bool haakjes_OK(string const &s)
{
   // Test of s een C++ opdracht bevat waarvan de haakjes in orde zijn.
   // Onderstelt dat s geen constante C-string of commentaar bevat.
   // Houdt enkel de openende haakjes op een stapel bij.
   // Een sluitend haakje moet overeenkomen met het overeenkomstig
   // openend haakje bovenaan de stapel, dat dan verwijderd wordt.
   stack<char> st;
   bool fout = false;
   int i=0;
   while(i<s.size() && !fout){
      if(s[i]=='(' || s[i]=='[' || s[i]=='{')
         st.push(s[i]);
      else if(s[i]==')' || s[i]==']'|| s[i]=='}')
         if(st.empty())
            fout = true;
         else if(st.top()=='(' && s[i]==')' ||
                 st.top()=='[' && s[i]==']' ||
                 st.top()=='{' && s[i]=='}')
            st.pop();
         else
            fout = true;
      i++;
   }
   return !fout && st.empty();
}

// KUIS DEZE CODE NOG OP:
// als we nu vragen om ook rekening te houden met de haakjes <>,
// zou je niet drie maar vier testen moeten doen
// - lelijk, langdradig en foutgevoelig!

// Oplossing:
// zorg dat je snel kan testen of een karakter een openend haakje is
// (steek deze dus in een set)
// en zorg dat je snel het bijhorende haakje van een sluitend haakje kan vinden
// (steek dus sluitende haakjes als key in een map,
// met het overeenkomstige openend haakje als value)
```

REEKS D

OGP in C++

Oefening 122

```
#include <iostream>
#include <fstream>
#include <string>  // nodig bij het inlezen van een breuk
#include <sstream> // vergelijk met Scanner(String) in Java
#include <cmath>   // abs berekent absolute waarde van een int (fabs is voor double)
#include <set>
using namespace std;

// Deze methode is ook nuttig buiten de Breuk-klasse.
// In Java zou je ervoor kiezen deze 'static' te maken
// (je kan immers niets buiten een klasse schrijven in Java);
// in C++ kan je een functie gerust extern schrijven.

// Merk op: de recursieve functie zou nog efficienter kunnen door gebruik
// te maken van een hulpfunctie die het recursieve werk op zich neemt
// zonder telkens te testen op <0, a<b en b==1.
int mijn_ggd(int a, int b){
    if(a < 0 || b < 0){
        return mijn_ggd(abs(a),abs(b));
    }
    if(a < b){
        return mijn_ggd(b,a);
    }
    if(b == 0){
        return a;
    }/*
    if(b == 1){
        return b;
    }*/
    return mijn_ggd(b,a%b);
}


class Breuk{
   private:
      int teller, noemer;

      void normaliseer() {
       if(noemer < 0) { noemer *= -1; teller *= -1; }
       int deler = mijn_ggd(teller,noemer);
       teller /= deler; noemer /= deler;
      }

   public:

      // voor deel 1
      Breuk():teller(0),noemer(1) {}
      Breuk(int t, int n=1):teller(t),noemer(n) { normaliseer(); }
      // alternatief: vervang beide constructoren door
      // public Breuk(int t=0, int n=1):teller(t),noemer(n) {normaliseer(); }

      // merk op: operator= en copyconstructor moet je niet schrijven
      //          want die bestaan al (en hun standaardwerking volstaat:
      //          er zijn immers geen pointers als dataleden)
```

```
      int get_teller() const { return teller; }

      // is geen lidfunctie; maar moet binnen de klassedefinitie staan
      // om friend gedeclareerd te kunnen worden.
      friend istream& operator>>(istream& in, Breuk & b);
      friend ostream& operator<<(ostream & uit, const Breuk & b);

      Breuk& operator+=(const Breuk& b);
      Breuk& operator-=(const Breuk& b);

      Breuk operator+(const Breuk& b) const;
      Breuk operator-(const Breuk& b) const;

      Breuk operator-() const;

      Breuk& operator++();
      Breuk operator++(int a);

      // voor deel 2
      bool operator==(const Breuk& b) const;
      bool operator!=(const Breuk& b) const;


      // voor deel 3
      Breuk operator+(int x) const;
      friend Breuk operator+(int x, const Breuk& b);

      bool operator<(const Breuk& b) const;
};

istream& operator>>(istream& in, Breuk & b) {
   string getalbeeld;
   in >> getalbeeld;
   stringstream ss; ss << getalbeeld;
   int positie = getalbeeld.find("/");
   if(positie != string::npos) {
      int t; char c; int n;
      ss >> t; ss >> c; ss >> n;
      if(c == '/' && !ss.fail()) b = Breuk(t,n);
      else b = Breuk();
   }
   else {  // misschien is er geen breukstreep, omdat je
           // enkel een geheel getal (dus met noemer = 1) opgaf
      int t; ss >> t;
      string overschot; ss >> overschot;
      if(overschot == "") b = Breuk(t);
      else b = Breuk();
   }
   return in;
}

ostream& operator<<(ostream & uit, const Breuk & b) {
   uit << b.teller ;
   if(b.noemer != 1) uit << "/" << b.noemer;
   return uit;
}

Breuk& Breuk::operator+=(const Breuk & b) {
   int deler = mijn_ggd(noemer,b.noemer);
   teller = b.noemer / deler * teller + noemer / deler * b.teller;
   noemer = noemer / deler * b.noemer;
```

```
   normaliseer();
   return *this;
}


Breuk& Breuk::operator-=(const Breuk & b) {
   operator+=(-b);
   return *this;
}

Breuk Breuk::operator-() const {
    return Breuk(-teller,noemer);
}

Breuk& Breuk::operator++(){
    teller += noemer;
    normaliseer();
    return *this;
}

Breuk Breuk::operator++(int a){
    Breuk temp(*this);
    teller += noemer;
    normaliseer();
    return temp;
}

bool Breuk::operator==(const Breuk& b) const {
   return teller == b.teller && noemer == b.noemer;
}

bool Breuk::operator!=(const Breuk & b) const {
   return !operator==(b);
}

Breuk Breuk::operator+(const Breuk & b) const {
   Breuk c = b;  // Beter: gebruik de copyconstructor; default-versie voldoet
                 // (geen pointers aanwezig als dataleden).
                 // Dus zet hier "Breuk c(a);"
                 // Dat is efficienter omdat je niet eerst een Breuk moet
                 // aanmaken met de defaultconstructor, om daarna weer
                 // dataleden te overschrijven.
                 // Idem voor andere operatoren.
   c += *this;
   return c;
}

Breuk Breuk::operator-(const Breuk& b) const {
   Breuk c(*this);
   c -= b;
   return c;
}

Breuk Breuk::operator+(int x) const{
    Breuk c(*this);
    x *= c.noemer;
    c.teller += x;
    return c;
}
// opmerking: mits een beetje basiswiskunde, kan je hier
// de kans op overflow sterk reduceren! Proberen maar....
```

```
Breuk operator+(int x, const Breuk& b){
    return b+x;
}

bool Breuk::operator<(const Breuk& b) const {
   return teller * b.noemer < noemer * b.teller;
}

bool is_stambreuk(const Breuk & a){
    return a.get_teller()==1;
}

////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////

void deel1(){
    Breuk a(2,5);
    Breuk b(1,-2);

    cout << a << " + " << b << " = ";
    cout << (a+b) << " [-1/10 ?]" << endl;

    cout << "De tegengestelde breuk van " << a << " is " << -a << " [-2/5 ?]." << endl;

    Breuk f = a + b;
    cout << "De som van " << a << " en " << b << " is " << f << " [-1/10 ?]" << endl;
    Breuk g(f);
    cout << "en dat is gelijk aan de breuk " << g <<" [-1/10 ?]."<<endl;

    cout << a << " - " << b << " = ";
    cout << (a-b) << " [9/10 ?]" << endl;
    cout << a << " += " << b << " geeft als resultaat dat de breuk " << a << " verandert
        in ";
    a += b;
    cout << a << " [-1/10 ?]" << endl;
    cout << a << " -= " << b << " geeft als resultaat dat de breuk " << a << " verandert
        in ";
    a -= b;
    cout << a << " [2/5 ?]" << endl;

    cout << "Ik verhoog nu de breuk a=" << a <<" met 2 eenheden; dan is a=";
    cout << ++(++a) << " [12/5 ?]" << endl;

    cout << "Na dit uitschrijven zal b=" << b++ <<" ook met een eenheid verhoogd zijn,
        nl. ";
    cout << "b=" << b << " [1/2 ?]" << endl;


    Breuk c(2,3);
    Breuk d(3,4);
    Breuk e(1,2);
    (c -= d) += e;
    cout << "Indien hier 5/12 staat, heb je de operatoren -= en += goed geschreven: " <<
        c << endl;
}

void deel2(){
    Breuk a(13,12);
    Breuk b(2);
    Breuk c;
    cout<<endl<<"Geef c op (onder de vorm teller/noemer (bvb 13/12) of teller (bvb 13) :
```

```
        ";
    cin >> c;
    if(a == c){
        cout << a << " is gelijk aan " << c <<endl;
    }
    if(a != c){
        cout << a << " is niet gelijk aan " << c <<endl;
    }
}

void deel3(){
    Breuk d(2,10);
    Breuk e(3);

    cout << d << " is stambreuk: "<<is_stambreuk(d)<<endl;

    Breuk f(3,4);
    cout << endl << "We starten van een breuk, en tellen er telkens een eenheid bij op: "
       << endl << endl;
    for(int i=0; i<10; i++){
        cout <<i<<" meer dan "<< f << " is " << (i+f) << " = " << (f+i) << endl;
    }

    cout << endl << "Al deze breuken in een verzameling: " << endl;
    set<Breuk> verz;
    for(int i=0; i<10; i++){
        verz.insert(i+f);
        verz.insert(f+i);
    }
    for(Breuk b : verz){
        cout << b << endl;
    }
}

int main(){
    deel1();
    deel2();
    deel3();
    return 0;
}
```

Oefening 123

```
#include <iostream>
#include <vector>

using namespace std;

template <typename T>
class Doos;

template <typename T>
class Schijf {
    friend ostream& operator<<(ostream& out, const Schijf<T> & schijf){
        schijf.schrijf(out);
        return out;
    }
    public:
        Schijf(): a(0){}
        Schijf(const Schijf<T>& schijf);
```

```
        Schijf<T>& operator=(const Schijf<T>&);
        ~Schijf<T>();

    private:
        Doos<T> *a;
};

template <typename T>
class Doos {
    public:
        // default constructor
        Doos();
        // copy constructor
        Doos(const Doos<T>&doos);
        // copy operator
        Doos<T>& operator=(const Doos<T>&);
        // destructor
        ~Doos();

        // constructor die b initialiseert
        Doos(const vector<T> &_b);

        // constructor die c initialiseert
        // als je hier de parameter van type
        // const Doos & maakt, dan ben je de
        // copyconstructor aan het schrijven;
        // dat willen we niet.
        // Dus onderscheid gemaakt door pointer
        // naar bron mee te geven
        Doos(const Doos<T>* doos);

        // constructor die d initialiseert
        Doos(Schijf<T> **_d);

    private:
        vector<T> b;
        Schijf<T> **d;
        Doos<T> *c;
};



//////////////////////////////////////////////////////
////////////////// DOOS

// default constructor
template<typename T>
Doos<T> :: Doos():b(4),c(0){
    cout << "default const" << endl;
    d=new Schijf<T>*[3];
    for(int i=0;i<3;i++) d[i]=0;
}

// copy constructor
template<typename T>
Doos<T> :: Doos(const Doos<T>& doos){
    b = doos.b;
    d = new Schijf<T>*[3];
    for(int i=0; i<3; i++){
        if(doos.d[i] != 0) d[i] = new Schijf<T>(*doos.d[i]);
        else d[i] = 0;
    }
```

```
    if(doos.c != 0){
        c = new Doos<T>(doos.c);
    }
    else c=0;
}

// copy operator
template<typename T>
Doos<T>& Doos<T> :: operator=(const Doos<T>&doos){
    if (this!=&doos) {
        b=doos.b;

        delete c;
        if(doos.c!=0){
            c = new Doos<T>(*(doos.c));
        }
        // Bij constructie van een Doos is er altijd voor gezorgd dat d
        // zeker en vast niet de null-pointer is, maar wel degelijk naar
        // een array van drie elementen wijst.
        // We testen er voor alle zekerheid toch nog even op
        // (misschien is er en-cours-de-route wat gewijzigd aan de situatie...):
        if(d==0){
            d = new Schijf*<T>[3];
            for(int i=0; i<3; i++){
                d[i]=0;
            }
        }
        else{
            for(int i=0;i<3;i++){
                delete d[i];
                d[i] = 0;
            }
        }
        // nu heb je een array van lengte 3, met 3 nullpointers in.
        // die moeten nu elk een nieuwe schijf toegewezen krijgen, als
        // de parameter 'doos' daar ook een schijf heeft.
        for(int i=0; i<3; i++){
            if (doos.d[i]!=0) d[i]=new Schijf<T>(*(doos.d[i]));
        }
    }
    return *this;
}


// constructor die b initialiseert
template<typename T>
Doos<T> ::  Doos(const vector<T> &_b): Doos(){
    cout << "vector const" << endl;
    b=_b;
}

// constructor die c initialiseert
template<typename T>
Doos<T> :: Doos(const Doos<T>* doos): Doos(){
    c = new Doos(*doos);
}

// constructor die d initialiseert
template<typename T>
Doos<T> :: Doos(Schijf<T> **_d):d(_d){
    b.resize(4);
    c=0;
```

```
}

// destructor
template<typename T>
Doos<T> :: ~Doos(){
    cout <<"destructor" << endl;
    delete c;
    for(int i=0;i<3;i++) delete(d[i]);
    delete []d;
}

////////////////////////////////////////////////////////
////////////////////// SCHIJF

template<typename T>
Schijf<T> :: Schijf(const Schijf<T>& schijf){
    if(schijf.a != 0)
        a=new Doos<T>(*(schijf.a));
    else
        a=0;
    // alternatief: de copyoperator hier gebruiken,
    // maar dan doet de compiler extra werk
    // dus niet doen!!
    // (ondanks dat het dan duplicated code is)
}

template<typename T>
Schijf<T> :: ~Schijf<T>(){
    delete a;
}

template <typename T>
Schijf<T>& Schijf<T> :: operator=(const Schijf<T>& schijf){
    if (this!=&schijf) {
        delete a;
        a=0;
        if (schijf.a!=0){
            a=new Doos<T>(*(schijf.a));
        }
    }
    return *this;
}
```

REEKS E

OGP: overerving

Oefening 124

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

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

class GekleurdeRechthoek : public Rechthoek {
 public:
    GekleurdeRechthoek();
    GekleurdeRechthoek(int, int);
    GekleurdeRechthoek(int, int, string);
    void print() const;
 protected:
    string kleur;
};

class Vierkant : public Rechthoek {
 public:
    Vierkant();
    Vierkant(int);
    void print() const;
};

class GekleurdVierkant : public Vierkant, public GekleurdeRechthoek {
 public:
    GekleurdVierkant();
    GekleurdVierkant(int);
    GekleurdVierkant(int, string);
    void print() const;
};

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() const {
    cout << "Rechthoek: " << breedte << " op " << hoogte
         << endl;
}
```

```
GekleurdeRechthoek::GekleurdeRechthoek(int h, int b, string kl)
   : Rechthoek(h,b), kleur(kl) {}
GekleurdeRechthoek::GekleurdeRechthoek(int h, int b)
   : GekleurdeRechthoek(h,b,"onbekend") {}
GekleurdeRechthoek::GekleurdeRechthoek() : kleur("onbekend") {}
void GekleurdeRechthoek::print() const {
    Rechthoek::print();
    cout << "  kleur: " << kleur << endl;
}

Vierkant::Vierkant(int zijde) : Rechthoek(zijde,zijde) {}
Vierkant::Vierkant() {}
void Vierkant::print() const {
    cout << "Vierkant: zijde " << hoogte << endl;
}

GekleurdVierkant::GekleurdVierkant(int zijde, string kl)
 : GekleurdeRechthoek(zijde,zijde,kl), Vierkant(zijde) {}
GekleurdVierkant::GekleurdVierkant() {}
GekleurdVierkant::GekleurdVierkant(int zijde)
 : GekleurdVierkant(zijde, "onbekend") {}
void GekleurdVierkant::print() const {
    Vierkant::print();
    cout << "  kleur: " << kleur << endl;
}

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

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

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

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

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

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

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

```
    GekleurdVierkant gv1;
    gv1.print();
    cout << "  oppervlakte: " << gv1.Vierkant::oppervlakte() << endl
         << "  omtrek: " << gv1.Vierkant::omtrek() << endl;

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

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

Oefening 125

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


template <class 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];
    out << " ]" << endl;
}


// eerst van vector<string>, daarna van vector<T> afleiden
template <class T>
class mijn_vector: public vector<T>{
    using vector<T>::vector;
  public:
    void verdubbel(bool herhaal_elk_element=false){ // per element; dus a b c wordt a a b
        b c c

        if(herhaal_elk_element){
            int lengte = this->size();
            this->resize(2*lengte);
            // zonder 'this->' krijg je de foutmelding
            //  "there are no arguments to 'resize' that depend on a template parameter,
            //   so a declaration of 'resize' must be available [-fpermissive]"
            for(int i=this->size()-1; i>0; i-=2){
                (*this)[i] = (*this)[i/2];
                (*this)[i-1] = (*this)[i/2];
            }
        }
        else{ // dan moet je element echt verdubbelen, dus *2
            for(int i=0; i<this->size(); i++){
```

```
                (*this)[i] = 2 * (*this)[i];
            }
        }
    }
};

// vervangt tweeklanken
void vervang(string & woord, const string & oud, const string & nieuw){
    int vindplaats = woord.find(oud);
    while(vindplaats != string::npos){
        string eerste_deel = woord.substr(0,vindplaats);
        string tweede_deel = woord.substr(vindplaats + oud.size());
        woord = eerste_deel + nieuw + tweede_deel;
        vindplaats = woord.find(oud,vindplaats + nieuw.size());
    }
}
// vervangt enkele klinkers, met uitsluiting van tweeklanken (die in 'verboden' map
    zitten)
void vervang(string & woord, const string & oud, const string & nieuw, const
    map<string,string> & verboden){
    int vindplaats = woord.find(oud);
    while(vindplaats != string::npos){
        string tweeklank = woord.substr(vindplaats,2);
        string tweeklank_voordien = "x";
        if(vindplaats!=0){
            tweeklank_voordien = woord.substr(vindplaats-1,2);
        }
        if(verboden.count(tweeklank) == 0 && verboden.count(tweeklank_voordien) == 0){
            string eerste_deel = woord.substr(0,vindplaats);
            string tweede_deel = woord.substr(vindplaats + oud.size());
            woord = eerste_deel + nieuw + tweede_deel;
            vindplaats = woord.find(oud,vindplaats + nieuw.size());
        }
        else{
            vindplaats = woord.find(oud,vindplaats + 5);
        }
    }
}
string p_taal(const string & s){
    map<string,string> verdubbeling1 =
        {{"aa","aapaa"},{"oe","oepoe"},{"ie","iepie"},{"ij","ijpij"},{"au","aupau"},{"uu","uupuu"},
                                        {"ei","eipei"},{"ou","oupou"},{"ui","uipui"},{"ee","eepee"},{"
    map<string,string> verdubbeling2 =
        {{"a","apa"},{"o","opo"},{"i","ipi"},{"y","ypy"},{"u","upu"},{"e","epe"}};
    string dubbel = s;
    map<string,string>::iterator it = verdubbeling1.begin();
    while(it!=verdubbeling1.end()){
        vervang(dubbel,it->first,it->second);
        it++;
    }
    it = verdubbeling2.begin();
    while(it!=verdubbeling2.end()){
        vervang(dubbel,it->first,it->second,verdubbeling1);
        it++;
    }
    return dubbel;
}

template<>
void mijn_vector<string>::verdubbel(bool herhaal_elk_element){
        if(herhaal_elk_element){
            resize(size()*2);
```

```
            for(int i=size()-1; i>0; i-=2){
                (*this)[i] = (*this)[i/2];
                (*this)[i-1] = (*this)[i/2];
            }
        }
        else{ // dan moet je element echt verdubbelen, voor een string: p-taal
            for(int i=0; i<this->size(); i++){
                (*this)[i] = p_taal((*this)[i]);
            }
        }
}

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;

    mijn_vector<string> s{"papageno","appelboom","poezenstaart"};
    cout<<endl<<"een vector met woorden: " << s;
    s.verdubbel();
    cout<<endl<<"na verdubbelen zonder parameter: " << s;
    s.verdubbel(true);
    cout<<endl<<"na verdubbelen met param true:   " << s;

    return 0;
}
```

Oefening 126

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

class Langeslang : 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;
    }
};

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();
    int s_c = c.size();
    resize(s_b+s_c);
    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;
}

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

int main(){
    Langeslang a;
    Langeslang b;
    Langeslang c;
    a.vul({1,2});
    b.vul({3,4,5});
    c.vul({6,7});
    cout<<"a: "<<a<<endl<<"b: "<<b<<endl<<"c: "<<c<<endl<<endl;

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

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

Oefening 127

```
#include <functional>
#include <iostream>
#include <vector>
#include <string>
```

```
using namespace std;

struct Persoon{
    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:
        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);
        }

};

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
      ([](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;

    return 0;

}
```

REEKS F

Excepties, OGP: dynamic binding en afsluiter

Oefening 128

```
// nodige aanpassingen:
// virtual,
// en gebruik van pointers (bij voorkeur unique_ptrs)

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

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

class GekleurdeRechthoek : public Rechthoek {
 public:
    GekleurdeRechthoek();
    GekleurdeRechthoek(int, int);
    GekleurdeRechthoek(int, int, string);
    virtual void print() const;
 protected:
    string kleur;
};

class Vierkant : public Rechthoek {
 public:
    Vierkant();
    Vierkant(int);
    void print() const;  // hier geen virtual!
};

class GekleurdVierkant : public Vierkant, public GekleurdeRechthoek {
 public:
    GekleurdVierkant();
    GekleurdVierkant(int);
    GekleurdVierkant(int, string);
    void print() const;
};

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() const {
    cout << "Rechthoek: " << breedte << " op " << hoogte
         << endl;
}

GekleurdeRechthoek::GekleurdeRechthoek(int h, int b, string kl)
   : Rechthoek(h,b), kleur(kl) {}
GekleurdeRechthoek::GekleurdeRechthoek(int h, int b)
   : GekleurdeRechthoek(h,b,"onbekend") {}
GekleurdeRechthoek::GekleurdeRechthoek() : kleur("onbekend") {}
void GekleurdeRechthoek::print() const {
    Rechthoek::print();
    cout << "  kleur: " << kleur << endl;
}

Vierkant::Vierkant(int zijde) : Rechthoek(zijde,zijde) {}
Vierkant::Vierkant() {}
void Vierkant::print() const {
    cout << "Vierkant: zijde " << hoogte << endl;
}

GekleurdVierkant::GekleurdVierkant(int zijde, string kl)
 : GekleurdeRechthoek(zijde,zijde,kl), Vierkant(zijde) {}
GekleurdVierkant::GekleurdVierkant() {}
GekleurdVierkant::GekleurdVierkant(int zijde)
 : GekleurdVierkant(zijde, "onbekend") {}
void GekleurdVierkant::print() const {
    Vierkant::print();
    cout << "  kleur: " << kleur << endl;
}

int main () {

   Rechthoek r2(4,6);
   GekleurdeRechthoek gr1;
   GekleurdeRechthoek gr3(6,9,"rood");
   Vierkant v2(10);

   vector<unique_ptr<Rechthoek>> v;
   v.push_back(make_unique<Rechthoek>(r2));
   v.push_back(make_unique<GekleurdeRechthoek>(gr1)); // type aanpassen aan gewenste
      type!!
   v.push_back(make_unique<GekleurdeRechthoek>(gr3));
   v.push_back(make_unique<Vierkant>(v2));

   for(auto& x : v){  // & nodig; anders maak je copies (en dat kan niet voor unique_ptr)!
        x->print();
        cout << "  oppervlakte: " << x->oppervlakte() << endl
             << "  omtrek: " << x->omtrek() << endl;
   }


   return 0;
}
```

Oefening 129

```
#include <stdexcept>
#include <iostream>
```

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


class bestand_niet_lang_genoeg : public invalid_argument{
    public:
        bestand_niet_lang_genoeg(const string & bestandsnaam, int nr):
           invalid_argument(bestandsnaam+" heeft geen "+to_string(nr)+" regels.") {}
};

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_genoeg(bestandsnaam,nr);
    }
    return zin;
}


int main(){

    vector<string>
       bestandsnamen({"niks","een","twee","drie","vier","vijf","zes","zeven","acht","negen","tien","
    vector<int> nrs({8,5,2,10,7,3,8,4,1,1,6,2,4});

    string bestanden_niet_gevonden = "";
    string bestanden_niet_lang_genoeg = "";
    string eerste_woorden = "";

    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_genoeg bnlg){
            bestanden_niet_lang_genoeg += string(bnlg.what());
        }
    }
```

```
    cout<<endl<<endl<<"BESTANDEN NIET GEVONDEN:"<<endl;
    cout<<bestanden_niet_gevonden;
    cout<<endl<<"BESTANDEN NIET LANG GENOEG:"<<endl;
    cout<<bestanden_niet_lang_genoeg<<endl;
    cout<<endl<<"BESTANDEN ZONDER STARTWOORD 'VERHAAL':"<<endl;
    cout<<"dit waren de woorden die er wel als eerste stonden:"<<endl<<endl;
    cout<<eerste_woorden<<endl<<endl;

    return 0;
}
```

Oefening 130

```
#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(){
}

// && om aan te geven dat je hier movet!
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){
    ifstream input(bestandsnaam);
    string soort;
    input >> soort;
    while (!input.fail() ){
       if (soort == "rechthoek"){
           double lengte, breedte;
           input >> lengte >> breedte;
           push_back(make_unique<Rechthoek>(Rechthoek(lengte,breedte)));
       }
       else if(soort == "vierkant"){
```

```
            double zijde;
            input >> zijde;
            push_back(make_unique<Vierkant>(Vierkant(zijde)));
        }
        else if(soort == "cirkel"){
            double straal;
            input >> straal;
            push_back(make_unique<Cirkel>(Cirkel(straal)));
        }
        else{
             string c;
             getline(input,c);  // om overschot in te lezen
        }
        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: ";
    cout<<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;
}
```