UNIVERSITEIT GENT

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

LABO Programmeren in C en C++
Oefeningenbundel
PARTIM C++

Leen Brouns
Helga Naessens
Wim Van den Breen

Opleiding Industrieel Ingenieur Informatica / Elektronica / Automatisering

september 2017

Doelstelling van de labo's C en C++

Lees voor de doelstellingen van de labo's C++ de oefeningenbundel PARTIM C.

Software

Zelfde opmerking voor de software.

Instellingen Dev-C++

Voor C++ -programma's komt er onder de compileropties de optie `-std=c++14` (met een kleine c) en de linkeropties vink je aan.

REEKS A

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

Oefening 101

Herneem oefening 1, maar dan in C++: schrijf op het scherm

```
Hello world!
10 9 8 7 6 5 4 3 2 1
START
```

Oefening 102

Hier komt de C++ -variant van oefening 2. Schrijf een programma dat alle (gehele) getallen van 0 tot en met 64 uitschrijft. Per regel komt zowel octale, decimale, als hexadecimale voorstelling van één getal. Zorg ervoor dat de getallen rechts gealligneerd zijn. Dit kan aan de hand van de manipulator `setw(aantal)` die ervoor zorgt dat de volgende operand van de uitschrijfoperator `<<` een vaste breedte van `aantal` karakters inneemt. Te vinden in de bibliotheek `iomanip`.

Oefening 103

Voorspel zonder computer waar de compiler fouten zal bespeuren. Verwijder (enkel) die regels uit de code, en voer uit.

```
const int AANT=10;
int getallen[AANT];
char letters[AANT];
string woorden[AANT];
cout<<"Geef "<<AANT<<" getallen, "
    <<"letters resp. woorden: ";
cin>>getallen;
cin>>letters;
cin>>woorden;
cout<<getallen<<endl;
cout<<letters<<endl;
cout<<woorden<<endl;
```

Oefening 104

Stel dat je een karakterteken wil omzetten naar een string (van lengte 1). Argumenteer waarom welk stukje code (niet) werkt.

```
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;
```

Oefening 105

1. Schrijf een functie `genereer_string(n)` die een random standaardstring genereert van lengte `n` (`n` is een gegeven geheel getal `>= 0`). De resulterende string bestaat dus uit een aaneenschakeling van n (al dan niet verschillende) random kleine letters.
2. Schrijf een procedure `vul_array_met_strings(tab, n, len)` die de array `tab` opvult met `n` random strings van lengte `len`. Je gebruikt hierbij uiteraard de voorgaande functie `geneer_string`.
3. Schrijf een procedure `bepaal_min_en_max(tab, n, min, max)` die in de gegeven array `tab` (die minstens `n` standaardstrings bevat) op zoek gaat naar de alfabetisch kleinste en grootste string (van de eerste `n`) en deze respectievelijk opslaat in de parameter `min` en `max`.
4. Test voorgaande procedures uit in een hoofdprogramma.

Oefening 106

Schrijf een functie `grootste(array,lengte)` die uit een gegeven array van gegeven lengte het grootste element teruggeeft. Het type van de elementen die in de array bewaard worden, is niet gekend. De grootte van een element wordt bepaald door de functie `grootte(elt)`, die dus zal geïmplementeerd moeten worden voor elk type waarvoor je de functie `grootste` oproept.

1. Schrijf een hoofdprogramma waarin je alvast volgende code schrijft:

   ```
   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;
   ```

   Zorg dat dit programma runt. De grootte van een woord wordt hier gedefinieerd als de lengte van het woord. Merk op dat we de functie `grootte(...)` niet meegeven aan de functie `grootste(...,...)`: we vragen hier dus geen functiepointers.
2. Vervolgens maak je ook een array van drie personen aan. Elke persoon is van type Persoon (definieer zelf de struct), en onthoudt zowel zijn naam als leeftijd (in jaren) en lengte (in meter). Schrijf een procedure `initialiseer(persoon,naam,leeftijd,lengte)` die de dataleden van een gegeven persoon initialiseert. Gebruik deze procedure om de drie personen in de array te initialiseren:
   De eerste persoon is Samuel, 12 jaar, 1m52.
   De tweede persoon is Jente, 22 jaar, 1m81.
   De derde persoon is Idris, 42 jaar, 1m73.
   Schrijf een procedure `print(persoon)` die de gegevens van een persoon uitschrijft op het scherm. (Test uit door een van de personen uit de tabel uit te schrijven.)
   Schrijf tenslotte de grootste persoon uit, als de grootte van een persoon bepaald wordt door zijn leeftijd. (Nadien pas je de code aan zodat de grootte van een persoon bepaald wordt door zijn/haar lengte respectievelijk de lengte van zijn/haar naam. Krijg je het verwachte resultaat?)

Oefening 107

Schrijf een procedure `schrijf(array,aantal,achterstevoren,tussenteken)` die een array van gehele getallen uitschrijft. De tweede parameter geeft de lengte van de array weer, de derde parameter bepaalt of het uitschrijven van achter naar voor dan wel op normale wijze moet gebeuren. De laatste parameter geeft het karakterteken mee dat tussen twee opeenvolgende getallen uitgeschreven wordt.

Declareer in het hoofdprogramma een array `t` met de getallen `1 3 5 7 9 11 13`. Roep daarna de procedure `schrijf` als volgt aan:

```
schrijf(t,7);
schrijf(t,7,true);
schrijf(t,7,false,'-');
schrijf(t,7,true,'-');
```

Dit zou de volgende output moeten produceren:

```
1 3 5 7 9 11 13
13 11 9 7 5 3 1
1-3-5-7-9-11-13
13-11-9-7-5-3-1
```

REEKS B

Kennismaking met unique pointers

Oefening 108

Gegeven onderstaande code.

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

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

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

int main(){

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

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

    return 0;
}
```

Je weet dat de regel code `s[i] = s[i+1];` het kopiëren van een string impliceert. Dat moeten we vermijden, want een string kan in principe heel groot zijn. Schrijf twee nieuwe procedures, die bij het onderstaande hoofdprogramma horen. We bewaren nu (unique) pointers in de array, zodat we bij het opschuiven van de elementen in de array enkel pointers moeten verleggen, en geen kopieën maken.

```
int main(){
    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;
}
```

Probeer ook eens het laatste element uit de array te verwijderen!

Oefening 109

In oefening 22 gebruikte je een functie als parameter voor een andere functie of procedure. In C (en C++) moest je die functie eerst expliciet een naam geven en implementeren. In C++11 kan het in één moeite: als het functievoorschrift klein genoeg is, maak je de functie on-the-fly aan. Doe dit om onderstaand hoofdprogramma aan de praat te krijgen: gebruik λ-functies op de plaats van `.......`. De functie `vul_array` moet je ook schrijven. Let op, het type van de vierde parameter is nu geen functiepointer, want je werkt in C++11/14 in plaats van C!

```
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,........);
schrijf("SOM:      ",c,GROOTTE);
vul_array(a,b,c,GROOTTE,........);
schrijf("PRODUCT:  ",c,GROOTTE);
vul_array(a,b,c,GROOTTE,........);
schrijf("VERSCHIL: ",c,GROOTTE);

return 0;
}
```

Oefening 110

Schrijf een C++-programma dat van alle (kleine) letters in het bestand `lord.txt` de frequenties uitschrijft. Om de frequenties bij te houden gebruik je een gewone array (nog geen vector); hoofdletters moet je niet tellen.

Oefening 111

Gegeven 2 tekstbestanden, `stationnetje.txt` en `paddestoel.txt`. Maak een derde bestand, `mix.txt`, waarin je de oneven regels van het eerste bestand laat afwisselen met de even regels van het tweede bestand. (Let wel: er zitten dus regels tussen die je *niet* in de mix terug zal vinden! Maak de mix zo lang als het kortste bestand.)

Breid dit daarna uit naar een mix van meerdere bestanden. Noem je programma `mix.cpp` en geef de bestandsnamen mee op de commandolijn. Het uitvoerbestand mag nog steeds `mix.txt` zijn. (Indien dit uitvoerbestand al bestaat, plak je de nieuwe output achteraan bij.)

De oproep kan er dan als volgt uitzien:

```
H:> mix stationnetje.txt paddestoel.txt khebdezonzienzakken.txt
```

REEKS C

Containers

Oefening 112

In de komende oefeningen gaan we containers vullen, raadplegen en veranderen. Om het effect hiervan duidelijk te zien, is het uiteraard aangeraden om containers eenvoudig te kunnen uitschrijven. Vandaar dat we in deze oefening zorgen voor code die de elementen van de verschillende containers (vector, stack, map, set,...) uitschrijft (hetzij op scherm, hetzij naar een bestand).

Een eerste poging staat hieronder. Deze procedure schrijft een vector van gehele getallen uit.

```
void schrijf(vector<int> v, int aantal){
    for(int i=0; i<aantal; i++){
        cout << v[i] <<" ";
    }
}
```

Hier valt echter redelijk wat op aan te merken. We verbeteren de code in stapjes.

1. Om te beginnen, *kapitale fout*: om de vector uit te schrijven, kopieer je eerst de hele rimram nog vóór je het eerste element op scherm uitschrijft. Zie theorieles: containers geef je altijd by reference door.
2. Bovendien kent een vector zijn eigen lengte, dus er is allicht geen reden om de parameter `aantal` mee door te geven.
3. Deze procedure werkt nu alleen voor vectoren met elementen van type `int`. Dus zou je het werk moeten herdoen voor een vector met elementen van type `double`. Dat is niet aangewezen, en kunnen we vermijden door het gebruiken van templates.

   Zorg dat je voor de vector met inhoud `1  2  3` de output `[ 1 | 2 | 3 ]` krijgt. Ga na dat je in dit geval ook vectoren van type `double`, `bool`, `string` en `char` kan uitschrijven.
4. Maak een struct `Persoon` aan (een persoon heeft een `naam` en een `leeftijd`) en probeer een vector, waar je één persoon aan toevoegt, uit te schrijven. Lukt dit? Wat mankeert er? Onderstaande code kan allicht helpen.

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

5. Stel dat je nu een vector van vectoren wil uitschrijven. Maak in het hoofdprogramma een vector `v` van vectoren aan, voeg aan `v` één vector toe die één element bevat, en schrijf uit. Wat merk je nu? Stel een oplossing voor en laat zien aan je lesgever.

Oefening 113

Ga voort op de vorige oefening. Voorzie code die volgende containers uitschrijft, telkens voor elementen van niet nader gespecifiëerd type. Je test telkens uit met een container waaraan je één (maximaal twee) elementen toevoegt.

- een set (schrijf elementen uit tussen accolades, met een komma ertussen)
- een stack (schrijf elementen uit onder elkaar; begin een nieuwe regel voor het eerste element)
- een map, waarbij je dus zowel type van sleutel als type van bijhorende data ongespecifiëerd laat (schrijf sleutels onder elkaar uit; elke sleutel wordt gevolgd door een pijltje en zijn bijhorende data)

Oefening 114

Je hebt in vorige oefeningen de uitschrijfoperator `<<` geïmplementeerd voor verschillende containers. Zet deze code (zonder het main-programma) in de headerfile met naam `containers.h`, en includeer deze header in het nieuwe `.cpp`-bestand. Gebruik indien nodig/nuttig.
Voor elke opdracht maak je een procedure aan, met naam `oefxx()`.

1. Maak een stack aan. Vul de stack met de woorden `"een"`, `"twee"`, `"drie"` en schrijf de stack daarna twee maal uit.
2. Maak een array aan van (oorspronkelijk lege) vectoren. De array heeft lengte AANTAL=5. De tweede vector vul je op met de woorden `aap`, `noot`, `mies`. Schrijf daarna de array uit.
3. Maak een (lege) vector aan van vectoren. Maak dan AANTAL=5 vectoren die hun lengte van bij de declaratie kennen: de `i`-de vector heeft lengte i (de eerste vector in het rijtje heeft dus lengte 0). De `i`-de vector is ook al meteen opgevuld met de getallen `10 20 ... 10*i`. Schrijf daarna de grote vector van achter naar voor uit; ook de 5 kleine vectoren schrijf je omgekeerd uit. Neem na elk van de kleine vectoren een nieuwe lijn.

Oefening 115

Maak een map aan, die karakters afbeeldt op verzamelingen met woorden. Daarna lees je (vanop klavier) een reeks woorden in die eindigt op `STOP`. De woorden worden volgens beginletter in de map opgeslagen. Daarna vraag je de gebruiker een letter, en schrijf je uit hoeveel ingelezen woorden er met deze letter beginnen. Je telt uiteraard alleen de verschillende woorden.

Oefening 116

Maak een vector aan van mappen. Elke map in die vector zal hetzelfde werk doen als de map uit vorige oefening, maar dan enkel voor de woorden van bepaalde lengte: de i-de map uit de vector houdt woorden van lengte i bij. (Het klopt dat de eerste en tweede map uit de vector dan redelijk nutteloos zijn, maar laten we daar even abstractie van maken.) Ga er, in een eerste versie, vanuit dat de woorden die je inleest maximaal lengte 70 hebben.
Om zeker te zijn dat de vector van mappen goed opgevuld is, schrijf je hem uit.
Vraag de gebruiker nu om een woord, en schrijf alle woorden uit die met dezelfde letter starten en even lang zijn. Wat uitgeschreven wordt moet niet noodzakelijk in alfabetische volgorde staan.

Test nu je programma met inputredirection. Je gebruikt daarvoor het bestand `bible_stop.txt`, dat voor de gelegenheid eindigt op het woord `STOP`.
Omdat dit nu niet te combineren is met invoer vanop het klavier (bij input redirection is het alles of niets!), mag je het zoekwoord hardcoderen: ga op zoek naar alle woorden in de bijbel die even lang zijn als `sinterklaas` en ook met een `s` beginnen. (Je kan ook `Sinterklaas` met hoofdletter opzoeken.) Merk op: de komma's en punten hangen nog aan de woorden vast, maar dat is niet erg.

Schrijf tenslotte een tweede versie: je maakt de vector van mappen maar net zolang als nodig. Dat wil zeggen dat je start bij lengte 10, en telkens je een woord tegenkomt dat toch langer is, resize je de vector van mappen (en kopieer/verplaats je de vorige inhoud; wees zuinig op acties!).

Oefening 117

Deze oefening vraagt geen code. Het enige wat je doet, is de declaratie neerschrijven van de container(s) die je gaat gebruiken als je deze oefening zou moeten oplossen. (Voel je na afwerken van de volledige reeks op containers nog de dringende nood aan meer oefening, dan heb je hier meteen wat opdrachten.)

Elke opdracht mag onafhankelijk van de vorige beschouwd worden. Je krijgt telkens een bestand, waarvoor je de opgesomde opdracht uitvoert.

(a) Geef het aantal verschillende woorden in het bestand.

(b) Schrijf de woorden uit vanaf het i-de woord tot en met het j-de woord, waarbij i en j gekend zijn voor je aan de opdracht begint.

(c) Schrijf het bestand achterstevoren uit.

(d) Geef de frequentie van één woord dat gekend is voor je aan de opdracht begint.

(e) Geef de vindplaats(en) van één woord dat gekend is voor je aan de opdracht begint.

(f) Geef de frequenties van alle woorden uit het bestand.

(g) Geef de vindplaats(en) van alle woorden uit het bestand.

Oefening 118

Dit is een vraag die ooit (onder licht andere bewoordingen) gesteld werd op een test. Gebruik enkel pen en papier, en verbeter daarna met de gegeven oplossing. (Wees ZEER KRITISCH! Dat doen wij ook op een test :-))
Gegeven een vector `v` met volgende declaratie.

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

1. Schrijf een procedure

   ```
   vul_in_zoveelste_map_beeld_van_sleutel_aan_met_set_van_drie
       (vector,index_van_map,sleutel,eerste,tweede,derde)
   ```

   Start je van een net-geïnitialiseerde vector, dan zal de oproep

   ```
   vul_in_zoveelste_map_beeld_van_sleutel_aan_met_set_van_drie
       (v,0,"noot","do","re","mi")
   ```

   ervoor zorgen dat de eerste map uit de vector het woord `noot` afbeeldt op een stack met één verzameling in. Die verzameling bevat de woorden `do`, `re` en `mi`.

2. Schrijf een logische functie

   ```
   zoveelste_map_beeldt_woord_af_op_stack_waarvan_bovenste_dit_element_bevat
       (vect,index,woord,element)}
   ```

   Deze functie bepaalt of de map op index `index` in de vector `vect` het woord `woord` afbeeldt op een stack, waarvan de bovenste set het woord `element` bevat.

Oefening 119

Gegeven een aantal woorden. Zorg dat ze op alfabetische volgorde staan (gebruik de juiste container), en haal er daarna om de drie woorden een woord uit. Dus het eerste, vierde, zevende,... woord uit de lijst verdwijnt. Controleer.

Oefening 120

Voor deze oefening haal je eerst de bestanden `nbible.txt` en `regelnummers.txt` af (zie intranet). Lees ook meteen dat laatste bestand, anders kan je de vraag niet volgen.

We verstopten in de bijbel een kort verhaaltje. Als je de regelnummers vermeld in het bestand met nummers vervangt door de tekst die in `nbible.txt` op die plaats te vinden is, kan je dit snel lezen. Opgelet! De bijbel is geen klein boekwerk. Vermijd twee maal doornemen van je bestand. Maar schrijf wél eerst regel nummer 18876 uit, en dan pas nummer 10000, anders klopt het verhaaltje niet. Gebruik de gepaste containers (eventueel meerdere!) om dit vlot te laten verlopen en bewaar uiteraard niet meer dan nodig.
LET OP! Eerst tekenen en op papier nagaan of je methode inderdaad tot een oplossing leidt. Pas als je redenering klopt als een bus, kan je aan code denken!

Oefening 121

Schrijf een logische functie `haakjes_OK(s)` die nagaat of de standaardstring `s` een C++-opdracht of -codefragment bevat waarin de eventuele haakjes (ronde, vierkante, accolades) goed staan. Onderstel voor de eenvoud dat het codefragment geen constante C-string (tussen dubbele aanhalingstekens) en geen commentaar bevat.
Zorg je er ook voor dat deze logische functie makkelijk uitbreidbaar is voor nog andere soorten haakjes (bijvoorbeeld `<>`)? Dat wil dus zeggen dat je geen `if / else if / else if / ... / else`-structuur gebruikt als het ook zonder kan.

REEKS D

OGP in C++

Een opmerking voor je aan deze reeks begint. Als je klassen schrijft, is het netjes om de klassedeclaratie in een aparte headerfile (met extensie `.h`) te schrijven, en de implementatie in een `.cpp`-bestand. De meeste editeeromgevingen zorgen dan automatisch voor het includeren en compileren van de juiste bestanden als je in een project werkt. Mogen we je echter met aandrang vragen om dit NIET UIT TE TESTEN in DevCpp? Daar maak je GEEN PROJECT aan; je werkt met losse bestanden (die dan wel in dezelfde map zitten), en includeert in het hoofdprogramma zelf de nodige klassen aan de hand van de directieve `#include "mijn_klasse.cpp"`. In het bestand `mijn_klasse.cpp` includeer je dan de headerfile `mijn_klasse.h`.

Oefening 122

Gegeven onderstaand hoofdprogramma, opgedeeld in drie stukken. Schrijf de klasse `Breuk` die hiervoor nodig is. (Haal de code uit het LaTeX-bestand dat je op Minerva vindt; dan zijn de kantlijnen netjes.) Enkele tips:

1. De (externe) functie `mijn_ggd` geeft de grootste gemene deler van twee gehele getallen.
2. Een breuk staat in zijn meest vereenvoudigde vorm als teller en noemer niet meer onderling deelbaar zijn. (Zijn ze dat nog wel, dan deel je hun grootste gemene deler uit om tot de meest vereenvoudigde vorm te komen.) Bewaar een breuk steeds in zijn vereenvoudigde vorm! (Schrijf een (hulp)lidfunctie `normaliseer()`.)
3. Een stambreuk is een breuk waarvan de teller gelijk is aan 1.

```
#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;

// zet deze functie in het bestand 'breuk.cpp'
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;
    }
    return mijn_ggd(b,a%b);
}


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;
```

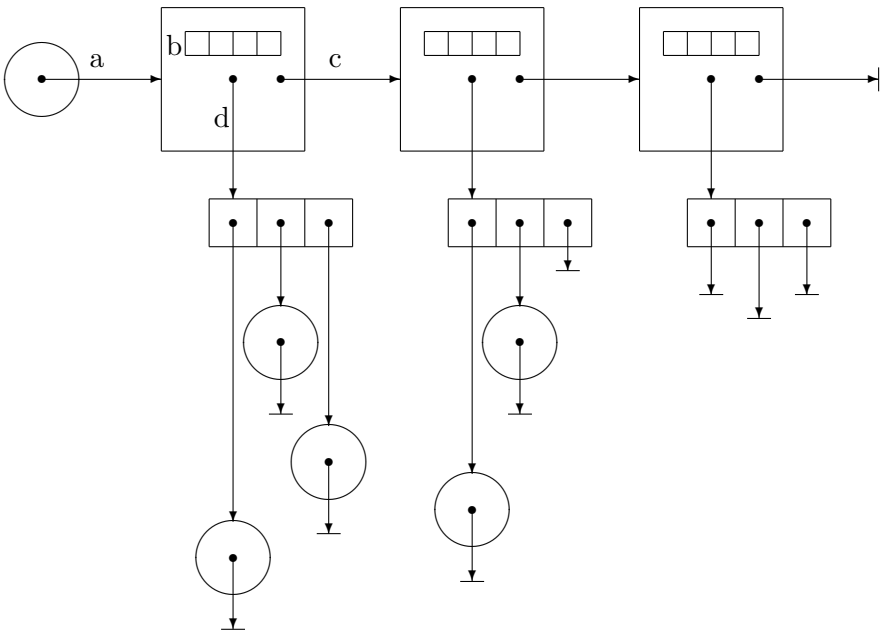
```
    for(int i=0; i<10; i++){
        verz.insert(i+f);
        verz.insert(f+i);
    }
    for(Breuk b : verz){
        cout << b << endl;
    }
}


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

Oefening 123

Gegeven: een schets met bijhorende legende.

1. een pijl stelt een *raw* pointer voor; de geheugenplaats waar hij bewaard wordt is het zwarte bolletje van waaruit de pijl vertrekt.
2. een pijl die aankomt op een dwars streepje, stelt een nullpointer voor.
3. een grote cirkel is een object van de klasse `Schijf`.
4. een groot vierkant is een object van de klasse `Doos`.
5. een rechthoek onderverdeeld in 4 vierkantjes is een vector van type `T` en lengte 4.
6. een rechthoek onderverdeeld in 3 vierkantjes is een array van lengte 3. Deze arrays zijn dynamisch aangemaakt, aan de hand van de pointer die naar deze array wijst.



Gevraagd: geef voor de klassen `Schijf` en `Doos` de dataleden. Implementeer daarna de default-constructor, destructor en copyconstructor. Schrijf ook de toekenningsoperator.

Voor de klasse `Doos` moet je nog het volgende weten: bij constructie zal het datalid van type vector onmiddellijk grootte 4 hebben. De pointer met naam `c` wijst voorlopig naar niets. De pointer met naam `d` wijst wel al naar een bestaande array; maar deze array bevat pointers die nog niet naar een nieuw object verwijzen.

Voor de klasse `Doos` tenslotte voorzie je ook drie constructoren die telkens één parameter nemen.
De eerste van die drie constructoren initialiseert het dataveld `b`.
De tweede van die drie constructoren initialiseert het dataveld `c` met een diepe kopie van de parameter.
De derde van die drie constructoren initialiseert het dataveld `d` waarbij de resources van de parameter overgenomen worden: er wordt geen diepe kopie gemaakt.