2 Unique pointers,λambdafuncties en containers

Oefening 109

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){
   for(int i = volgnr; i < aantal-1; i++){
       s[i] = s[i+1];
   }
   s[aantal-1] = "";  //laatste element leeg maken
}

int main(){

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

    schrijf(namen,5);
    verwijder(namen,5,1);
    schrijf(namen,5); //alle namen tonen

    return 0;
}
```

Test uit wat het programma doet. Het laatste element is leeg, maar wordt nog steeds getoond (dat is de bedoeling!).

Probleem: De code `s[i] = s[i+1];` zal de string **kopiëren**. Dat moeten we vermijden, want een string kan in principe heel groot zijn.

Opdracht: 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[]={...}; //vul zelf deze array aan met 5 unieke pointers
    schrijf(pnamen,5);
    verwijder(pnamen,5,1);
    schrijf(pnamen,5);

    return 0;
}
```

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

Oefening 110

In oefening 25 werd een functie als parameter doorgegeven voor een andere functie of procedure. In C (en C++) moet je die functie eerst expliciet een naam geven en implementeren.

In C++11 kan het in één moeite: met λ-functies maak je de functie 'on-the-fly' aan.

Pas onderstaand hoofdprogramma aan:

- schrijf de procedure `schrijf`.
- schrijf de procedure `vul_array`. .
  Let op, het type van de vierde parameter is nu geen functiepointer, want je werkt in C++11/14 in plaats van C!
- vervolledig de aanroep van de methode `vul_array`: vervang `.....` door een λ-functie.

```
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 111

In de volgende oefeningen gaan we containers vullen, raadplegen en veranderen. Het is handig om methodes te hebben die de elementen van de verschillende containers (vector, stack, map, set,...) uitschrijven (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. Een vector kent zijn eigen lengte. De tweede parameter is dus overbodig.

3. Deze procedure werkt nu alleen voor vectoren met elementen van type `int`. Pas dit aan met templates.
   Controleer dat je met de procedure 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 toont hoe je de uitschrijfoperator `<<` kan implementeren (door *overloading*) voor een willekeurige `out`-stream en een variabele van het type `Persoon`. Dit komt later uitgebreid aan bod.

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

   Verander de procedure `void schrijf(...)` naar uitschrijfoperator `<<` en test dit uit in het hoofdprogramma.
5. Maak in het hoofdprogramma een vector `v` van vectoren aan, voeg aan `v` één vector toe die één element bevat. Schrijf de vector uit met `cout<<` .

Oefening 112

Ga verder op de vorige oefening. Implementeer de uitschrijfoperator `<<` voor volgende containers met 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 113

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.

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 114

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 115

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.)
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.

Oefening 116

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 117

Dit is een vraag die ooit (onder licht andere bewoordingen) gebruikt werd voor 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_i_de_map_stack_bij_sleutel_aan_met_set_van_drie_strings
       (vect,i,sleutel,str1,str2,str3)
   ```

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

   ```
   vul_in_i_de_map_stack_bij_sleutel_aan_met_set_van_drie_strings
       (v,1,"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

   ```
   i_de_map_beeldt_woord_af_op_stack_waarvan_bovenste_set_dit_element_bevat
       (vect,i,woord,element)
   ```

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

Oefening 118

Maak een collectie van `n` willekeurig gekozen gehele getallen die kleiner zijn dan `2*n` . Kies vooraf een willekeurig waarde voor `n` (bijvoorbeeld tussen 20 en 30). Bovendien moeten de getallen geordend zijn (gebruik de juiste container).
Haal er daarna om de drie getallen één getal uit. Dus het eerste, vierde, zevende,... getal verdwijnt uit de collectie. Controleer.

Oefening 119

Voor deze oefening haal je eerst de bestanden `nbible.txt` en `regelnummers.txt` af (zie Minerva). 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!

- Heb je alles in het hoofdprogramma uitgewerkt? Pas dit aan zodat je drie procedures toevoegt, die elk een afgebakend deel van de oplossing uitwerken.
- Lees je altijd de volledige bijbel in? Van zodra je regel **29454** (de laatste die je nodig hebt) ingelezen hebt, kan je het inlezen beter stoppen.
  Pas dit aan.