C++ifPCwhileLinuxdiffintFirefoxboolpublicprivateWindows

fileg++UNIXVisualOSFedoralscharpsopenclosegetputarray

Maccpelsecoutmain RedHatmkdireditorstructDebianSublime

classusingcmathChrometruecpSUSEnamespaceenuminclude

doublecdGNUdokillobjectcompilermorelinkiostreamcinnotlpr

WWWmakefilegrepiomanipgeditfloatGNOMEfstreamrmfalse

andorCode::Blockstopforstringe-mailmanchmodKDEUbuntu

Programmeermethoden

Walter Kosters en Jonathan Vis

week 1: 6–10 september 2021 Introductie

www.liacs.leidenuniv.nl/~kosterswa/pm/

Het college wordt gegeven door dr. W.A. (Walter) Kosters en dr. J.K. (Jonathan) Vis.

Werkcolleges worden verzorgd door studentassistenten. Donderdagen: Miguel Blom, Isaac Braam, Sem Kluiver, Steven van Popele, Thijs Snelleman en Jakob Wuhrer; vrijdagen: Iris Leijten, Xander Lenstra en Ali Esat Özbay.

Anderen geven PM-Python voor Bioinformatica en Informatica & Economie; en voor Natuur/Sterrenkunde.

Er zijn verschillende soorten activiteiten:

colleges live; zie ook streams, video's en oudere video's van 15–20 minuten; in de twee eerste weken: Wiskunde en Informatica apart

$\leqslant 75$ mensen

werkcolleges in computerzalen Snellius 302, . . . , handig: laptop mee; iedereen welkom op donderdag of vrijdag

vragenuren in Snellius, direct na werkcollege

live-rooms via Kaltura ???

Eerste week: 6–10 september 2021; laatste week: 13–17 december. Geen activiteiten in de week van 25–29 oktober.

www.liacs.leidenuniv.nl/~kosterswa/pm/schema.php

maandag 6 september	di	wo	donderdag 9 september	vrijdag 10 september
16:15–18:00 Gorlaeus zaal 2 $\leqslant 75$ college 1 Wiskunde			9:15–11:00 PieterDeLaCourt zaal SC01 $\leqslant 75$: A-K college 1 Informatica	14:15–16:00 Snellius 302,306 werkcollege 1 Wiskunde
			14:15–16:00 Snellius 302,306,. . . werkcollege 1 Informatica	16:15–. . . Snellius 412,407 vragenuur Wiskunde

voor streams, video's en oude (korte) video's, zie:

www.liacs.leidenuniv.nl/~kosterswa/pm/videos.php

maandag 13 september	di	wo	donderdag 16 september	vrijdag 17 september
14:15–16:00 Gorlaeus zaal 2 $\leqslant$ 75 college 2 Wiskunde			9:15–11:00 Gorlaeus zaal 2 $\leqslant$ 75: L-Z college 2 Informatica	14:15–16:00 Snellius 302,306 werkcollege 2 Wiskunde
			14:15–. . . Snellius 302,306,. . . werkcollege 2, vragenuur Informatica	16:15–. . . Snellius 412,407 vragenuur Wiskunde

Vanaf derde week samen college in Gorlaeus zaal 1/. . . , donderdagen 11:15–13:00.

Snellius

Gorlaeus

Iedereen gebruikt zijn/haar ULCN-account. Daarmee kun je ook WiFi gebruiken in universiteitsgebouwen, via `eduroam` of `leidenuniv`. Zet een "forward" voor e-mails!

Je krijgt daarmee ook een *home directory* ("homedir") in het universitaire UNIX-systeem. In de computerzalen van het Snellius kun je daar via Linux mee werken.
Hoe kom je vanaf huis bij je UNIX-files? Antwoord: "ssh-en" en "scp-en" naar

`a.einstein@sshgw.leidenuniv.nl`

(en verder) als je `a.einstein` bent. Zie later.

Sluw: laptop, USB, eigen e-mail, de cloud, . . .

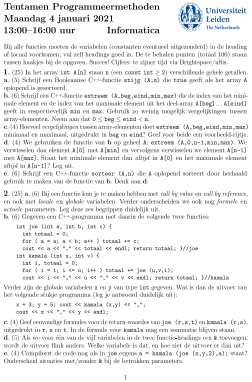
Het schriftelijk tentamen is op

- woensdag 5 januari 2022, 14:15–17:15 uur
- hertentamen: donderdag 31 maart 2022, 14:15–17:15 uur

Zie de website voor oude tentamens en uitwerkingen.

Tijdig aanmelden via uSis! En Brightspace (tegelijk) . . .

4–1–2021 $\longrightarrow$



Tentamen Programmeermethoden
Maandag 4 januari 2021
13:00–16:00 uur **Informatica**

Universiteit Leiden
The Netherlands

Bij alle functies moeten de variabelen (constanten eventueel uitgezonderd) in de heading of bocad voorkomen; vul zelf headings goed in. De te behalen punten (totaal 100) staan tussen haakjes bij de opgaven. Succes! Cijfers: te zijner tijd via (Brightspace/)uSis.

1. (25) In het array `int A[n]` staan `n` (een `const int` ≥ 2) verschillende gehele getallen.
a. (5) Schrijf een Booleanse C++-functie `stijg (A,n)` die `true` geeft als het array `A` oplopend is gesorteerd.
b. (6) Schrijf een C++-functie `extreem (A,beg,eind,min_max)` die de index van het minimale element en de index van het maximale element uit het deel-array `A[beg]` . . `A[eind]` geeft in respectievelijk `min` en `max`. Gebruik zo weinig mogelijk vergelijkingen tussen array-elementen. Neem aan dat $0 \leq$ `beg` $\leq$ `eind` $<$ `n`.
c. (4) Hoeveel vergelijkingen tussen array-elementen doet `extreem (A,beg,eind,min_max)` minimaal en maximaal, uitgedrukt in `beg` en `eind`? Geef voor beide een voorbeeld-cijfje.
d. (4) We gebruiken de functie van **b** op gebied `A: extreem (A,0,n-1,min_max)`. We vervolgen dan element `A[0]` met `A[min]` en vervolgens verwisselen we element `A[n-1]` met `A[max]`. Staat het minimale element dan altijd in `A[0]` en het maximale element altijd in `A[n-1]`? Leg uit.
e. (6) Schrijf een C++-functie `sorteer (A,n)` die `A` oplopend sorteert door herhaald gebruik te maken van de functie van **b**. Denk aan **d**.

2. (25) **a**. (6) Bij een functie kun je te maken hebben met *call by value* en *call by reference*, en ook met *locale* en *globale* variabelen. Verder onderscheiden we ook nog *formele* en *actuele* parameters. Leg deze zes begrippen duidelijk uit.
b. (6) Gegeven een C++-programma met daarin de volgende twee functies:

```
int joe (int a, int b, int c) {
  int totaal = 0;
  for ( a = a; a < b; a++ ) totaal += c;
  cout << a << "," << totaal << endl; return totaal; }//joe
int kamala (int x, int y) {
  int i, totaal = 0;
  for ( i = 1; i <= u; i++ ) totaal += joe (u,v,1);
  cout << i << "," << y << "," << v << endl; return totaal; }//kamala
```

Verder zijn de globale variabelen `x` en `y` van type `int` gegeven. Wat is dan de uitvoer van het volgende stukje programma (log `y` antwoord duidelijk uit).

```
x = 3; y = 5; cout << kamala (x,y) << ",";
cout << x << "," << y << endl;
```

c. (4) Geef eenvoudige formules voor de return-waarden van `joe (r,s,t)` en `kamala (r,s)`, uitgedrukt in `r`, `s` en `t`. In de formule voor `kamala` mag een sommatie blijven staan.
d. (6) Als we voor één van de vijf variabelen in de twee functie-headings een `&` toevoegen, wordt de uitvoer flink anders. Welke variabele is dat, en hoe ziet de uitvoer er dan uit?
e. (4) Compileert de code nog als in `joe` ergens `a = kamala (joe (x,y,2),s);` staat? Onderscheid situaties met/zonder `&` bij de betrokken parameters.

1

3. (25) Gegeven is een `n` bij `n` (beide `const int` > 0; ze hoeven bij deze opgave niet te worden doorgegeven als parameter) array `B` met hoofdletters en kleine letters. We spelen een soort *Stratego* waarbij de letters de rangen aangeven. De hoofdletters en kleine letters spelen tegen elkaar. Hiernaast een voorbeeld met `n` $=$ `4` en `n` $=$ `5`.

D	w	j	c	O
b	r	h	O	X
B	d	C	S	s
E	a	e	X	v

a. (7) Schrijf een C++-functie `int balanz (B)` die de balanz tussen hoofdletters en kleine letters berekent: het verschil tussen beide totaalwaarden. Hierbij telkens `A` en `a` voor 0, `B` en `b` voor 1, etc. In het voorbeeld is dit $107 - 99 = 8$ (in het voorbeeld van de hoofdletters).
b. (8) Nu kunnen hoofdletters horizontaal of verticaal direct aangrenzende kleine letters slaan (en vice versa) indien hun waarde echt hoger is. In het voorbeeld kan de hoofdletter `E` op positie (1,2) alleen kleine letter `j` op positie (0,2) slaan. Schrijf een Booleanse C++-functie `slaat (B,i,j,p,q)` die controleert of er op positie (i,j) een hoofdletter staat die op positie (p,q) kan slaan. Neem aan dat $0 \leq$ `i,p` $<$ `n` en $0 \leq$ `j,q` $<$ `n`.
c. (10) Schrijf een C++-functie `int bereik (B,p,q)` die berekent welke hoofdletter de beste positie heeft. Dat betekent dat deze de aangrenzende kleine letters kan slaan met de hoogste (opgetelde) totaalwaarde. In het voorbeeld is dat `E` op positie (3,3), want deze kan 6 op positie (3,2) en 8 op positie (3,4) slaan voor een totaal van $4 + 21 = 25$. Gebruik de functie van **b**. De positie van de beste hoofdletter moet worden doorgegeven via `p` en `q`, en de bijbehorende totaalwaarde wordt gereturnd. Als er meerdere de beste zijn: één van deze. Als geen enkele hoofdletter kan slaan: geef 0 terug, ook in `p` en `q`.

4. (25) Gegeven is het volgende type:

```
class object { public: int info; object* onder; object* rechts; };
```

Met behulp hiervan kan een structuur met twee zijen met objecten worden opgebouwd, bestaande uit enkele met een getal, en twee pointers; zie onderstaand plaatje. Voor beide zijen geldt: de `rechts`-pointer wijst naar het volgende object er rechts naast (als laatste een `nullptr`). In de bovenste rij wijst de `onder`-pointer naar het er onder gelegen object, in de onderste rij terug naar boven. Een voorbeeld, met `ingang` van type `object*`:

a. (6) Schrijf een C++-functie `erbij (ingang,getal1,getal2)` die twee nieuwe objecten, onder elkaar, met `getal1` en `getal2` erin vooraan de lijst met ingang `ingang` toevoegt.
b. (6) Schrijf een C++-functie `verwijder (ingang)` die het eerste tweetal objecten (onder elkaar) uit de lijst met ingang `ingang` netjes verwijdert, indien deze bestaan, en beide getallen > 0 zijn. (In het voorbeeld: de objecten met 12 en 94.)
c. (4) Schrijf een C++-functie `som (ingang)` die de waarde in het eerste object (als dat er is) ophoogt met de waarde in het object er schuin rechts onder (indien aanwezig), zulks het resultaat door 8 deelbaar is. In het voorbeeld: $12 \rightarrow 12 + 20 = 32$.
d. (3) In de functies bij **a**, **b** en **c** staat in de heading een pointer. Deze heb je call by value of call by reference doorgegeven (met een `&`). Maakt het voor de werking van deze functies verschil uit of die `&` erbij staat? Mag het, moet het? Leg duidelijk uit.
e. (6) Schrijf een C++-functie `sorteer (ingang)` die de getallen in de bovenste rij oplopend sorteert met een aangepaste *bubblesort*: wissel zolang het nodig is naburige getallen (en met de pointers!) die verkeerd staan.

2

Er moeten vier programmeeropgaven gemaakt worden. **Als** ze alle voldoende zijn (hooguit één 5) **en** het tentamen voldoende ($\geqslant 5.5$) is gemaakt, krijg je zes studiepunten:

$$\text{Eindcijfer} = \frac{2 \times \text{Schriftelijk} + \frac{\text{Op}_1 + \text{Op}_2 + \text{Op}_3 + \text{Op}_4}{4}}{3}$$

(afgerond naar het dichtstbijzijnde element uit de verzameling $\{1, 2, 3, 4, 5,\ 6, 6.5, 7, 7.5, 8, 8.5, 9, 9.5, 10\}$).

Heb je nog deelresultaten uit voorgaande jaren? Ga langs bij de docent!

Cijfers: Brightspace (meld je aan = "enroll").

- in tweetallen maken
- wel overleggen en om hulp vragen, niet kopiëren;
  zie website voor huisregels en richtlijnen, bijvoorbeeld
  voor aanvullen, . . .
- `op tijd` inleveren: deadlines (−1 per week te laat)
- deels maken tijdens werkcolleges, deels thuis
- vragenuren!

- Opgave 1: maandag 27 september 2021, 17:00 uur
- Opgave 2: maandag 18 oktober 2021, 17:00 uur
- Opgave 3: maandag 15 november 2021, 17:00 uur
- Opgave 4: maandag 13 december 2021, 17:00 uur

Stuur per tweetal een fatsoenlijke e-mail met C++-code, dus één attachment, naar:

`pm@liacs.leidenuniv.nl`

En het geprinte verslag in de doos bij Snellius-159.

Programma 2021 — Eerste helft

week	onderwerp	boek	"dictaat"
6–10 sep	Introductie	1	1,2
13–17 sep	Types		3.1/3,3.9, op1/5
20–24 sep	Controlestructuren	2	3.4/5,op6/10
27 sep–1 okt	Functies, files	3,4, 12.1/2	3.6/7,op11/17
4–8 okt	idem, vervolg		
11–15 okt	idem, Life	6,7.1	3.11,op18/25
18–22 okt	OOP, arrays	5	3.8,op26/30
...			

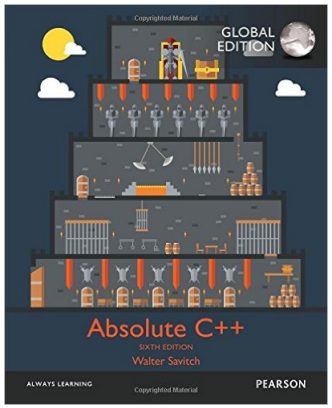
op = "papieren" opgaven van website ("Handouts"); zelf maken, antwoorden: zie website.

In rood: de weken met een deadline op de maandag erna.

We maken gebruik van het volgende boek:

W. Savitch
Absolute C++
sixth edition
Addison-Wesley, 2016

Oudere drukken zijn ook goed (ouderejaars!). En er zijn vele andere boeken.

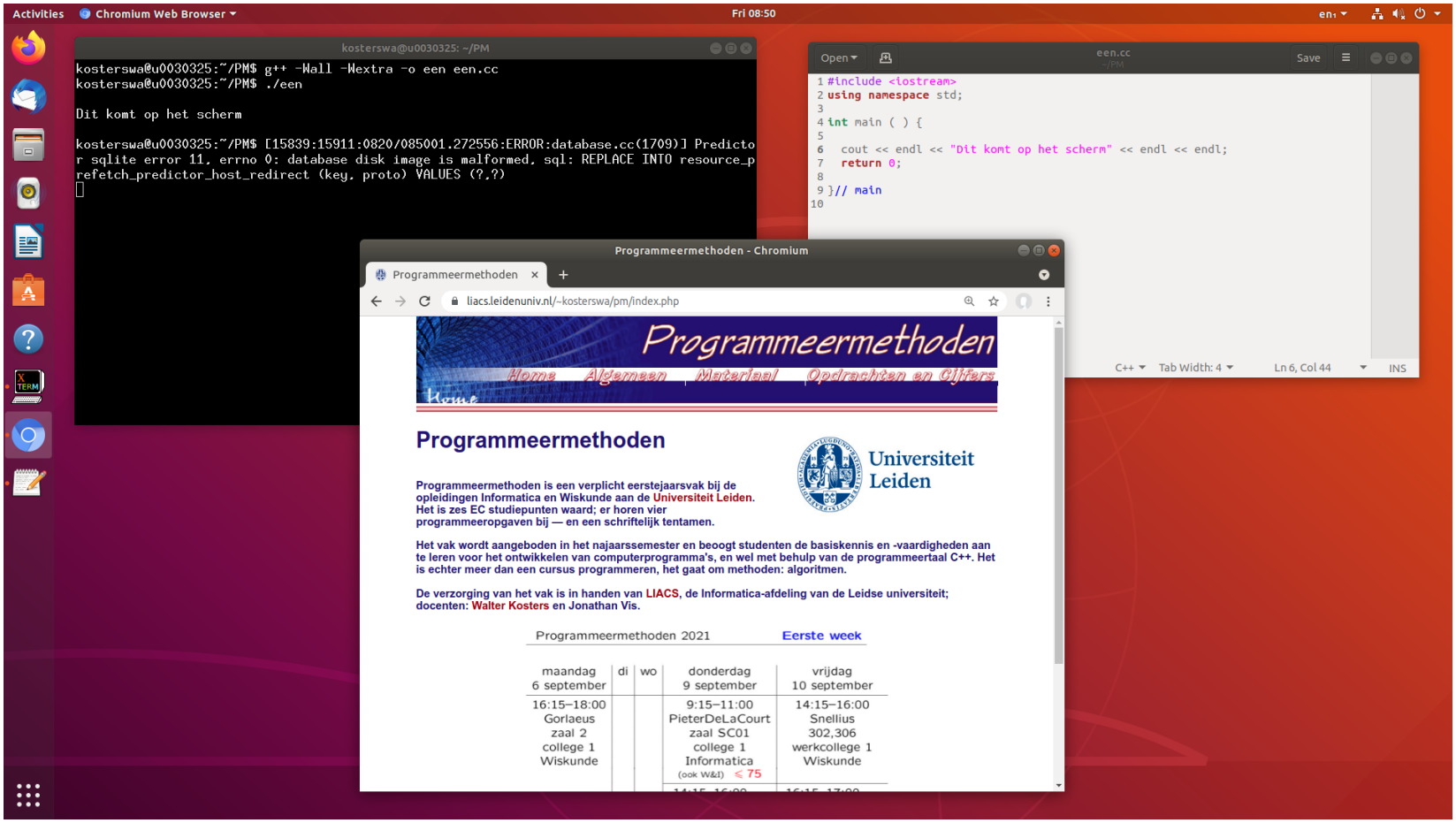


En het "dictaat" (hier) en de sheets (daar).

Naast Microsoft Windows kunnen PCs ook draaien onder Linux, een operating systeem (OS) uit de UNIX-wereld (vergelijk Android).

Met een Ubuntu Live-CD/DVD/USB start je PC meteen op in GNOME, een grafische windows-omgeving bovenop Linux.
Beter: dual boot.
Let er op dat je op een verstandige plaats (USB/e-mail) moet saven.
Nog beter: WSL (zie straks).

Je kunt ook "gratis" distributies als SUSE, Fedora, Debian, ... gebruiken — maar dan moet je er meer vanaf weten.


Activities
Chromium Web Browser
Fri 08:50
kosterswa@u0030325: ~/PM
kosterswa@u0030325:~/PM$ g++ -Wall -Wextra -o een een.cc
kosterswa@u0030325:~/PM$ ./een
Dit komt op het scherm
kosterswa@u0030325:~/PM$ [15839:15911:0820/085001.272556:ERROR:database.cc(1709)] Predictor sqlite error 11, errno 0: database disk image is malformed, sql: REPLACE INTO resource_prefetch_predictor_host_redirect (key, proto) VALUES (?,?)
Open
een.cc
~/PM
Save
1 #include <iostream>
2 using namespace std;
3
4 int main ( ) {
5
6 cout << endl << "Dit komt op het scherm" << endl << endl;
7 return 0;
8
9 }// main
10
C++
Tab Width: 4
Ln 6, Col 44
INS
Programmeermethoden - Chromium
Programmeermethoden
liacs.leidenuniv.nl/~kosterswa/pm/index.php
Programmeermethoden
Home
Algemeen
Materiaal
Opdrachten en Cijfers
Programmeermethoden
Universiteit Leiden
Programmeermethoden is een verplicht eerstejaarsvak bij de opleidingen Informatica en Wiskunde aan de Universiteit Leiden. Het is zes EC studiepunten waard; er horen vier programmeeropgaven bij — en een schriftelijk tentamen.
Het vak wordt aangeboden in het najaarssemester en beoogt studenten de basiskennis en -vaardigheden aan te leren voor het ontwikkelen van computerprogramma's, en wel met behulp van de programmeertaal C++. Het is echter meer dan een cursus programmeren, het gaat om methoden: algoritmen.
De verzorging van het vak is in handen van LIACS, de Informatica-afdeling van de Leidse universiteit; docenten: Walter Kosters en Jonathan Vis.
Programmeermethoden 2021
Eerste week
maandag 6 september
di
wo
donderdag 9 september
vrijdag 10 september
16:15–18:00 Gorlaeus zaal 2 college 1 Wiskunde
9:15–11:00 PieterDeLaCourt zaal SC01 college 1 Informatica (ook W&I) ⩽ 75
14:15–16:00 Snellius 302,306 werkcollege 1 Wiskunde

Als je thuis gratis C++ wilt doen, zijn de mogelijkheden:

- Windows: Code::Blocks, zie straks
- Mac: zie hier

  Mac-video
- Linux: haal Ubuntu van www.ubuntu.org
  huidige versie: 21.04 (in computerzalen: 18.04)
  gebruik editor `gedit` en compiler `g++`

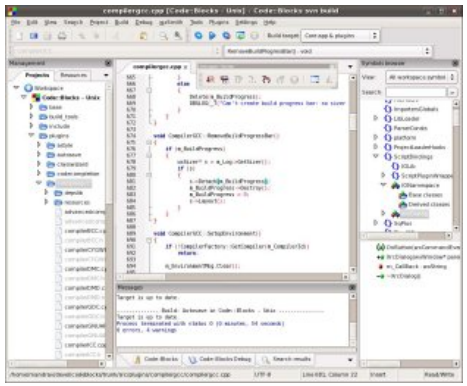
  Linux-video
- Windows10: installeer Windows Subsystem for Linux (WSL) en editor Sublime Text

We gebruiken in het begin vooral Code::Blocks in Windows, daarnaast ook `g++` en `gedit` in Linux.

video

- installeer Code::Blocks van `www.codeblocks.org`
  file: `codeblocks-20.03mingw-setup.exe`

- zet "warnings" aan!



- edit, compileer en run
  `helloworld.cc` (let op `.cc`)

- behandel een eventuele lastige firewall

Een werkend C++-programma maken gaat als volgt:

1. Tik in een editor C++-code, het "bron-programma".
2. Compileer (en link) dit met een compiler. Deze vertaalt C++-code naar machinetaal.
   Eventuele fouten: "compile-time-fouten" / "syntax-errors": "`inklude`" in plaats van "`include`".
3. Draai (= run) deze executable vanuit het OS.
   Eventuele fouten: "run-time-fouten": deel door 0.

Herhaal deze cyclus zo vaak als nodig.
In Code::Blocks vinden alle stappen plaats in een Graphical User Interface (GUI).

(Er zijn ook nog logische fouten.)

Een eerste C++-programma, `helloworld.cc` geheten:

```
#include <iostream>
using namespace std;
int main ( ) {
  cout << "Dag allemaal!" << endl;
  return 0;
}//main
```

Dit programma zet alleen een tekstje op het beeldscherm.

Let op de — vooral voor mensen nuttige — layout.
En op hoofdletters en kleine letters.

Een tweede C++-programma (met syntax-highlighting):

```
1  // dit is een simpel programma
2  #include <iostream>
3  using namespace std;
4
5  int main ( ) {
6    int getal = 42;   // een variabele declareren
7                      // en initialiseren
8    cout << "Geef een geheel getal .. ";
9    cin >> getal;
10   cout << "Kwadraat is: "
11        << getal * getal << endl;
12   return 0;
13 }//main
```

Nog een C++-programma

```
// Dit is een regel met commentaar ...
#include <iostream> // moet er altijd bij
using namespace std;
const double pie = 3.14159; // een constante (of cmath)
int main ( ) {
  int straal;  // straal van de cirkel, geheel getal
  cout << "Geef straal, daarna Enter .. ";
  cin >> straal;
  if ( straal > 0 )
    cout << "Oppervlakte "
         << pie * straal * straal << endl;

  else
    cout << "Niet zo negatief ..." << endl;
  cout << "Einde van dit programma." << endl;
  return 0;
}//main
```

Nog een C++-programma

```
// Dit is een regel met commentaar ...
#include <iostream> // moet er altijd bij
using namespace std;
const double pie = 3.14159; // een constante (of cmath)
int main ( ) {
  int straal;  // straal van de cirkel, geheel getal
  cout << "Geef straal, daarna Enter .. ";
  cin >> straal;
  if ( straal > 0 ) {        // accolades nodig!
    cout << "Oppervlakte ";
    cout << pie * straal * straal << endl;
  }//if
  else
    cout << "Niet zo negatief ..." << endl;
  cout << "Einde van dit programma." << endl;
  return 0;
}//main
```

Eerste programmeeropgave

Programmeermethoden 2021
Eerste programmeeropgave: abc-formule

De *eerste* programmeeropgave van het vak **Programmeermethoden** in het najaar van 2021 heet *abc-formule*; zie ook het **eerste**, **tweede** en **derde** werkcollege.

Deze opgave probeert te bepalen of iemand geschikt is voor een studie aan de universiteit: er is immers geen loting. Daartoe moeten enkele vragen beantwoord worden; zo moet de kandidaat weten op welke dag hij/zij geboren is. En als je de abc-formule niet kunt gebruiken, is een beta-studie misschien niet verstandig.

Om te beginnen moet de gebruiker zijn/haar geboortejaar als geheel getal invoeren, en daarna de geboortemaand, ook als geheel getal. Vervolgens voert hij/zij de geboortedag in, wederom als geheel getal. Het programma berekent dan de leeftijd van de gebruiker, zowel in aantal jaren als in maanden (bijvoorbeeld: 10 jaar en 3 maanden; 123 maanden); beide worden op het beeldscherm getoond. De leeftijd in maanden wordt analoog aan die in jaren bepaald (als je op de 31ste geboren bent, wordt je iedere maand een maand ouder, maar je bent niet zo vaak "maandig" — dat ben je namelijk alleen op iedere 31ste). Jarige en maandige gebruikers worden gefeliciteerd. Aangenomen mag worden dat het programma op de peildatum *27 september 2021* draait (gebruik `const`; **liefhebbers** mogen met `ctime` de echte huidige dag opvragen en gebruiken). Let op: het programma moet in principe ook op andere peildata vanaf heden tot 2100 correct werken!
Gebruikers jonger dan 10 jaar (de 10-de verjaardag nog niet gevierd) of ouder dan 100 jaar (dus 101-ste verjaardag reeds gevierd) worden meteen geweigerd. Als uit het geboortejaar direct al duidelijk is dat het met de leeftijd niets gaat worden, hoeven de vragen naar maand en/of dag niet gesteld te worden. Maar soms biedt pas de dag uitsluitsel!
Nu moet de gebruiker zijn/haar geboortedag (zondag, maandag, ..., zaterdag) weten. Als deze fout is, wordt men meteen "verwijderd", en stopt het programma. Het antwoord moet met *één letter* (de eerste letter van de dag; geen cijfer dus) worden gegeven, bijvoorbeeld w voor woensdag. In het geval van d/z wordt nog om de tweede letter gevraagd.
Het is *niet* de bedoeling `ctime` te gebruiken om deze dag uit te rekenen. Het programma moet een zelf bedachte berekening bevatten om deze dag te bepalen! Gebruik bijvoorbeeld dat 1 januari 1901 op een dinsdag viel. Gebruik *niet* het **Doomsday algoritme** (zie ook **hier**), en ook *niet* allerlei ingewikkelde formules. Voor de periode 1901–2099 geldt dat een jaar een schrikkeljaar is precies dan als het jaartal door 4 deelbaar is.

De echte test bestaat uit enkele vragen. Mensen van 30 jaar of ouder worden hierbij twee maal "netter" aangesproken dan jongeren. Splits de C++-code in het programma niet onnodig vaak!
Er wordt gekeken of de aanstaande student de abc-formule kan gebruiken. Wiskundig inzicht is namelijk vereist voor een beta-studie. Mocht dat niet zo zijn, wordt er getest hoe het met de kunst- of literatuurkennis staat.

Een kwadratische vergelijking heeft 0, 1 of 2 reële oplossingen, die we kunnen vinden met behulp van de **abc-formule**. Het programma genereert een willekeurige kwadratische vergelijking van de vorm $a\,x^2 + b\,x + c = 0$ en toont deze "netjes" op het scherm (de exponent mag er uit zien als `x^2`). Hierbij geldt dat *a*, *b* en *c* gehele getallen zijn (in absolute waarde maximaal 1000000), waarbij *a* groter dan 0 is. De student wordt gevraagd hoeveel reële oplossingen zij/hij denkt dat deze vergelijking heeft: 0, 1 of 2. In alle gevallen worden de oplossingen van de vergelijking (als die er zijn) op het scherm afgedrukt. Liefhebbers mogen met complexe getallen werken.
Voor *a*, *b* en *c* moeten `int`'s gebruikt worden. Omdat de "discriminant" te groot kan zijn voor een `int`, moet een `double` gebruikt worden in de berekeningen. Voor worteltrekken kan `y = sqrt (x)` worden gebruikt; sqrt zit in `cmath`. Voor het fabriceren van willekeurige gehele getallen moet gebruik worden gemaakt van de random-generator uit C++. Gebruik bijvoorbeeld `x = rand ( ) % 20;` om een "willekeurig" getal tussen 0 en 19 (grenzen inbegrepen) in de `int` variabele `x` te krijgen. Zet bovenaan in main: srand (42); of `srand (jaar);` (nadat `jaar` een waarde heeft gekregen), om de random-generator eenmalig te initialiseren. In plaats van 42 mag ook een ander getal staan — of zelfs, voor liefhebbers, de tijd. En soms is hiervoor `#include <cstdlib>` nodig, helemaal bovenaan het programma.

Is het antwoord goed, dan wordt de kandidaat tot een exacte studie toegelaten, en stopt het programma. Anders wordt één meerkeuzevraag (Aa/Bb/Cc/Dd) over kunst of literatuur gesteld, die uitsluitsel biedt over de toelating tot een alpha-studie. Als het daar ook mis gaat, is men helaas niet geschikt voor een universitaire studie. Gebruikers tot of tot en met (kies zelf) 30 jaar krijgen hier een andere vraag dan de oudere gebruikers

— maar bij beiden is "hetzelfde" antwoord, bijvoorbeeld steeds B, goed. Of het antwoord goed of fout is, het juiste antwoord wordt steeds op het scherm afgedrukt.

Opmerkingen

Als de gebruiker een niet bestaande maand invoert, bijvoorbeeld −8, of een jaartal als 4242 (in de toekomst dus), stopt het programma met de mededeling dat dit niet kan (gebruik `return 1;`). Evenzo voor een niet bestaande dag, bijvoorbeeld 31 april of 42 december. We nemen aan dat de gebruiker zo vriendelijk is verder geen fouten te maken bij het invoeren van gegevens: hij/zij voert niet al te gekke getallen of letters in, etcetera. Vanzelfsprekend worden hem/haar wel duidelijke vragen gepresenteerd.

Elk programma moet bij het "runnen" aan het begin op het beeldscherm laten zien wie de makers zijn, wat hun jaar van aankomst, studierichting en studentnummer is, welke opgave het is, wat de gebruiker te wachten en te doen staat, de datum waarop het programma gemaakt is, enzovoorts. Dit noemen we het **infoblokje**. Probeer dit er netjes uit te laten zien. Maak geen al te complexe kaders eromheen; gebruik liefst alleen de eerste 128 gewone karakters.
Bovenaan het programma (in de C++-code dus) staat uiteraard commentaar, waarin een aantal van deze elementen ook weer terugkomen, maar dan meer gericht op programmeurs, bijvoorbeeld de naam van de gebruikte compiler.

Denk aan het gebruik van lege regels, inspringen, commentaar, constanten, enzovoorts. Bovenaan het programma dient zoals gezegd commentaar over het programma te staan, speciaal bestemd voor andere *programmeurs* (en nakijkers), bijvoorbeeld kort wat het programma doet, en welke compiler gebruikt is: gebruikers van het programma vinden dat laatste niet interessant. Het infoblokje moet tijdens het "runnen" van het programma op het scherm komen, en is bestemd voor *gebruikers* van het programma. Lees ook eens over **richtlijnen** bij het maken van programmeeropgaven, en bestudeer de **huisregels**. Er hoeft geen gebruik van functies, arrays en het while- en for-statement gemaakt te worden. Alleen de headerfiles `iostream` mag en moet gebruikt worden — en eventueel `ctime` voor liefhebbers; en misschien `cstdlib` voor het gebruik van de random-generator. Ruwe indicatie voor de lengte van het C++-programma: 200 regels (300 mag ook wel).
Letters moeten als `char` worden ingelezen, dus *niet* met strings, die mogen namelijk niet gebruikt wordem.

Uiterste inleverdatum: **maandag 27 september 2021, 17:00 uur**.
De manier van inleveren (één exemplaar per koppel, dat — ter herinnering — uit maximaal twee personen bestaat) is als volgt. Digitaal de C++-code **inleveren**: stuur een email naar `pm@liacs.leidenuniv.nl`; en een print van het verslag in de doos bij kamer 159 van het Snellius. Stuur geen executable's, LaTeX-files of PDF-files, lever alleen één C++-file digitaal in! Noem deze bij voorkeur `bidenharris1.cc`, dit voor de eerste opdracht van het duo Harris / Biden. De laatst voor de deadline ingeleverde versie wordt nagekeken.
Tip: maak een nette e-mail, met een korte maar zinnige tekst als inhoud, en de C++-file als attachment. Overal duidelijk datum en namen van de twee makers vermelden, in het bijzonder als commentaar in de eerste regels van de C++-code. Lees bij het **derde werkcollege** hoe het verslag eruit moet zien. Zijn spaties/tabs goed verwerkt?

Te gebruiken compiler: als hij maar C++ vertaalt; het programma moet in principe zowel op een Linux-machine (met g++) als onder Windows met Code::Blocks draaien. Test dus zo mogelijk op beide systemen! Normering: (consequente) layout 2; commentaar 2; infoblokje 1; verslag 1; werking 4. Eventuele **aanvullingen en verbeteringen**: lees de WWW-bladzijde die je nu ziet: `www.liacs.leidenuniv.nl/~kosterswa/pm/op1pm.php`.
Recente aanvullingen en/of wijzigingen staan in **rood**.

www.liacs.leidenuniv.nl/~kosterswa/pm/op1pm.php

Voor de eerste programmeeropgave moet je onder meer, voor een gegeven datum, de dag van de week uitrekenen. Bijvoorbeeld: 1–1–1901 $\longrightarrow$ dinsdag.

Hoe zou je dat uit je hoofd doen?

Wanneer is een jaar een schrikkeljaar?

www.liacs.leidenuniv.nl/~kosterswa/pm/op1pm.php

Verslag in LaTeX! En hoeveel oplossingen heeft $x^2 + 1 = 0$?

Niet gebruiken: Doomsday-algoritme 5–9: zondag
4–4, 6–6, 8–8, 10–10, 12–12, "I work from 9–5 in a 7–11".

In een Linux windows-omgeving zoals KDE of GNOME start je een of meer terminals: windows waarin je tekst-georiënteerde opdrachten kunt geven. Daarin tik je in:

`gedit een.cc &`	edit je eerste C++-programma; open hiertoe een "edit-window"
`g++ -Wall -o een een.cc`	compileer `een.cc` naar `een`
`./een`	run de executable `een`
`ls -lrt`	overzicht van je files (%)

De ampersand `&` zorgt er voor dat je in het oorspronkelijke window ook kunt doorwerken. En (%) levert zoiets als:

```
-rw-r--r-- 1 kosterswa domain users   124 Sep 6 12:52 een.cc
-rwx------ 1 kosterswa domain users 11049 Sep 6 12:53 een
```

- donderdag 14:15–. . . (Informatici)
- vrijdag 14:15–. . . (Wiskundigen)
- met aansluitend vragenuur (vrijdag: zalen 174 en 407)
- op PC's in computerzalen Snellius (laptops in 313)
- www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc1.php
- doel: compiler, Hello world, int, opgave, . . .

- Hello world voor C++
- boek en “dictaat”
- ULCN-account
- software voor thuis/laptop
- de eerste programmeeropgave
- www.liacs.leidenuniv.nl/~kosterswa/pm/
- huiswerk: Savitch Hoofdstuk 1; dictaat 1 en 2

Programmeermethoden

Types

Walter Kosters en Jonathan Vis

week 2: 13–17 september 2021

www.liacs.leidenuniv.nl/~kosterswa/pm/

maandag 13 september	di	wo	donderdag 16 september	vrijdag 17 september
14:15–16:00 Gorlaeus zaal 2 $\leqslant$ 75 college 2 Wiskunde			9:15–11:00 Gorlaeus zaal 2 $\leqslant$ 75: L-Z college 2 Informatica	14:15–16:00 Snellius 302,306 werkcollege 2 Wiskunde
			14:15–. . . Snellius 302,306,. . . werkcollege 2, vragenuur Informatica	16:15–. . . Snellius 412,407 vragenuur Wiskunde

Maandag 13 september, vanaf 16:15 uur, is er een "informeel" vragenuur in de computerzalen 302 en 306.

maandag 20 september	di	wo	donderdag 23 september	vr
16:15–. . . Snellius 302,306 “informeel” vragenuur			11:15–13:00 Gorlaeus zaal 1 $\leqslant 75$ college 3 “iedereen” A-G	
			14:15–. . . Snellius 302,306,. . . werkcollege 3, vragenuur Informatica	

In verband met het weekend van De Leidsche Flesch is het werkcollege/vragenuur van vrijdag 24 september verplaatst naar maandag 27 september, 14:15–. . . ; Snellius 174/312.

Eerste programmeeropgave

Programmeermethoden 2021
Eerste programmeeropgave: abc-formule

De *eerste* programmeeropgave van het vak Programmeermethoden in het najaar van 2021 heet *abc-formule*; zie ook het eerste, tweede en derde werkcollege.

Deze opgave probeert te bepalen of iemand geschikt is voor een studie aan de universiteit: er is immers geen loting. Daartoe moeten enkele vragen beantwoord worden; zo moet de kandidaat weten op welke dag hij/zij geboren is. En als je de abc-formule niet kunt gebruiken, is een beta-studie misschien niet verstandig.

Om te beginnen moet de gebruiker zijn/haar geboortejaar als geheel getal invoeren, en daarna de geboortemaand, ook als geheel getal. Vervolgens voert hij/zij de geboortedag in, wederom als geheel getal. Het programma berekent dan de leeftijd van de gebruiker, zowel in aantal jaren als in maanden (bijvoorbeeld: 10 jaar en 3 maanden; 123 maanden); beide worden op het beeldscherm getoond. De leeftijd in maanden wordt analoog aan die in jaren bepaald (als je op de 31ste geboren bent, wordt je iedere maand een maand ouder, maar je bent niet zo vaak "maandig" — dat ben je namelijk alleen op iedere 31ste). Jarige en maandige gebruikers worden gefeliciteerd. Aangenomen mag worden dat het programma op de peildatum *27 september 2021* draait (gebruik `const`; liefhebbers mogen met `ctime` de echte huidige dag opvragen en gebruiken). Let op: het programma moet in principe ook op andere peildata vanaf heden tot 2100 correct werken!
Gebruikers jonger dan 10 jaar (de 10-de verjaardag nog niet gevierd) of ouder dan 100 jaar (dus 101-ste verjaardag reeds gevierd) worden meteen geweigerd. Als uit het geboortejaar direct al duidelijk is dat het met de leeftijd niets gaat worden, hoeven de vragen naar maand en/of dag niet gesteld te worden. Maar soms biedt pas de dag uitsluitsel!
Nu moet de gebruiker zijn/haar geboortedag (zondag, maandag, ..., zaterdag) weten. Als deze fout is, wordt men meteen "verwijderd", en stopt het programma. Het antwoord moet met *één letter* (de eerste letter van de dag; geen cijfer dus) worden gegeven, bijvoorbeeld w voor woensdag. In het geval van d/z wordt nog om de tweede letter gevraagd.
Het is *niet* de bedoeling `ctime` te gebruiken om deze dag uit te rekenen. Het programma moet een zelf bedachte berekening bevatten om deze dag te bepalen! Gebruik bijvoorbeeld dat 1 januari 1901 op een dinsdag viel. Gebruik *niet* het Doomsday algoritme (zie ook hier), en ook *niet* allerlei ingewikkelde formules. Voor de periode 1901–2099 geldt dat een jaar een schrikkeljaar is precies dan als het jaartal door 4 deelbaar is.

De echte test bestaat uit enkele vragen. Mensen van 30 jaar of ouder worden hierbij twee maal "netter" aangesproken dan jongeren. Splits de C++-code in het programma niet onnodig vaak!
Er wordt gekeken of de aanstaande student de abc-formule kan gebruiken. Wiskundig inzicht is namelijk vereist voor een beta-studie. Mocht dat niet zo zijn, wordt er getest hoe het met de kunst- of literatuurkennis staat.

Een kwadratische vergelijking heeft 0, 1 of 2 reële oplossingen, die we kunnen vinden met behulp van de abc-formule. Het programma genereert een willekeurige kwadratische vergelijking van de vorm $a\,x^2 + b\,x + c = 0$ en toont deze "netjes" op het scherm (de exponent mag er uit zien als `x^2`). Hierbij geldt dat a, b en c gehele getallen zijn (in absolute waarde maximaal 1000000), waarbij a groter dan 0 is. De student wordt gevraagd hoeveel reële oplossingen zij/hij denkt dat deze vergelijking heeft: 0, 1 of 2. In alle gevallen worden de oplossingen van de vergelijking (als die er zijn) op het scherm afgedrukt. Liefhebbers mogen met complexe getallen werken.
Voor *a*, *b* en *c* moeten `int`'s gebruikt worden. Omdat de "discriminant" te groot kan zijn voor een `int`, moet een `double` gebruikt worden in de berekeningen. Voor worteltrekken kan `y = sqrt (x)` worden gebruikt; sqrt zit in `cmath`. Voor het fabriceren van willekeurige gehele getallen moet gebruik worden gemaakt van de random-generator uit C++. Gebruik bijvoorbeeld `x = rand ( ) % 20;` om een "willekeurig" getal tussen 0 en 19 (grenzen inbegrepen) in de `int` variabele `x` te krijgen. Zet bovenaan in main: `srand (42);` of `srand (jaar);` (nadat `jaar` een waarde heeft gekregen), om de random-generator eenmalig te initialiseren. In plaats van 42 mag ook een ander getal staan — of zelfs, voor liefhebbers, de tijd. En soms is hiervoor `#include <cstdlib>` nodig, helemaal bovenaan het programma.

Is het antwoord goed, dan wordt de kandidaat tot een exacte studie toegelaten, en stopt het programma. Anders wordt één meerkeuzevraag (Aa/Bb/Cc/Dd) over kunst of literatuur gesteld, die uitsluitsel biedt over de toelating tot een alpha-studie. Als het daar ook mis gaat, is men helaas niet geschikt voor een universitaire studie. Gebruikers tot of tot en met (kies zelf) 30 jaar krijgen hier een andere vraag dan de oudere gebruikers

— maar bij beiden is "hetzelfde" antwoord, bijvoorbeeld steeds B, goed. Of het antwoord goed of fout is, het juiste antwoord wordt steeds op het scherm afgedrukt.

Opmerkingen

Als de gebruiker een niet bestaande maand invoert, bijvoorbeeld −8, of een jaartal als 4242 (in de toekomst dus), stopt het programma met de mededeling dat dit niet kan (gebruik `return 1;`). Evenzo voor een niet bestaande dag, bijvoorbeeld 31 april of 42 december. We nemen aan dat de gebruiker zo vriendelijk is verder geen fouten te maken bij het invoeren van gegevens: hij/zij voert niet al te gekke getallen of letters in, etcetera. Vanzelfsprekend worden hem/haar wel duidelijke vragen gepresenteerd.

Elk programma moet bij het "runnen" aan het begin op het beeldscherm laten zien wie de makers zijn, wat hun jaar van aankomst, studierichting en studentnummer is, welke opgave het is, wat de gebruiker te wachten en te doen staat, de datum waarop het programma gemaakt is, enzovoorts. Dit noemen we het infoblokje.
Probeer dit er netjes uit te laten zien. Maak geen al te complexe kaders eromheen; gebruik liefst alleen de eerste 128 gewone karakters.
Bovenaan het programma (in de C++-code dus) staat uiteraard commentaar, waarin een aantal van deze elementen ook weer terugkomen, maar dan meer gericht op programmeurs, bijvoorbeeld de naam van de gebruikte compiler.

Denk aan het gebruik van lege regels, inspringen, commentaar, constanten, enzovoorts. Bovenaan het programma dient zoals gezegd commentaar over het programma te staan, speciaal bestemd voor andere *programmeurs* (en nakijkers), bijvoorbeeld kort wat het programma doet, en welke compiler gebruikt is: gebruikers van het programma vinden dat laatste niet interessant. Het infoblokje moet tijdens het "runnen" van het programma op het scherm komen, en is bestemd voor *gebruikers* van het programma. Lees ook eens over richtlijnen bij het maken van programmeeropgaven, en bestudeer de huisregels. Er hoeft geen gebruik van functies, arrays en het while- en for-statement gemaakt te worden. Alleen de headerfiles `iostream` mag en moet gebruikt worden — en eventueel `ctime` voor liefhebbers; en misschien `cstdlib` voor het gebruik van de random-generator. Ruwe indicatie voor de lengte van het C++-programma: 200 regels (300 mag ook wel). Letters moeten als `char` worden ingelezen, dus *niet* met strings, die mogen namelijk niet gebruikt worden.

Uiterste inleverdatum: maandag 27 september 2021, 17:00 uur.
De manier van inleveren (één exemplaar per koppel, dat — ter herinnering — uit maximaal twee personen bestaat) is als volgt. Digitaal de C++-code inleveren: stuur een email naar `pm@liacs.leidenuniv.nl`; en een print van het verslag in de doos bij kamer 159 van het Snellius. Stuur geen executable's, LaTeX-files of PDF-files, lever alleen één C++-file digitaal in! Noem deze bij voorkeur `bidenharris1.cc`, dit voor de eerste opdracht van het duo Harris / Biden. De laatst voor de deadline ingeleverde versie wordt nagekeken.
Tip: maak een nette e-mail, met een korte maar zinnige tekst als inhoud, en de C++-file als attachment. Overal duidelijk datum en namen van de twee makers vermelden, in het bijzonder als commentaar in de eerste regels van de C++-code. Lees bij het derde werkcollege hoe het verslag eruit moet zien. Zijn spaties/tabs goed verwerkt?

Te gebruiken compiler: als hij maar C++ vertaalt; het programma moet in principe zowel op een Linux-machine (met `g++`) als onder Windows met Code::Blocks draaien. Test dus zo mogelijk op beide systemen! Normering: (consequente) layout 2; commentaar 2; infoblokje 1; verslag 1; werking 4. Eventuele aanvullingen en verbeteringen: lees de WWW-bladzijde die je nu ziet: `www.liacs.leidenuniv.nl/~kosterswa/pm/op1pm.php`. Recente aanvullingen en/of wijzigingen staan in rood.

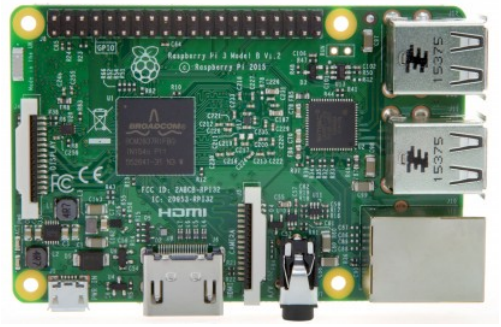
www.liacs.leidenuniv.nl/~kosterswa/pm/op1pm.php

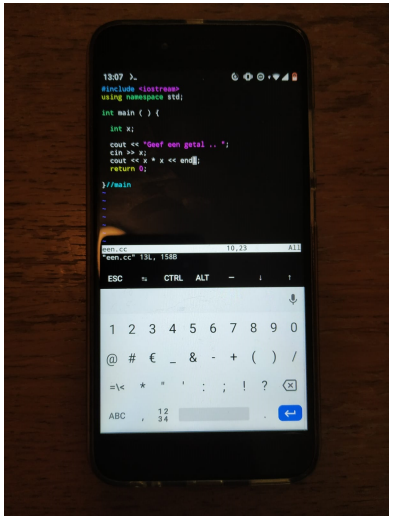
Voor we aan de types beginnen, eerst nog een "uitgebreide" stoomcursus UNIX en wat meer over C++.

Mac: www.liacs.leidenuniv.nl/~kosterswa/pm/macdoc.php

Code::Blocks, Linux (WSL), . . . :
www.liacs.leidenuniv.nl/~kosterswa/pm/videos.php

Smartphone (Termux)?
Of een Raspberry Pi?





De belangrijkste commando's zijn:

`ls`	overzicht files in directory (= map)
`lpr`	file printen
`more`	file op beeldscherm
`rm`	file verwijderen
`cp`	file kopiëren
`mv`	file verplaatsen
`cd`	van directory veranderen
`mkdir`	maak een nieuwe directory
`chmod`	rechten bij files regelen
`man`	hulp

Enkele voorbeelden:

```
ls -lrt ; cp een twee ; cd Abc ; chmod 644 * ; man ls
```

(En file `a.cc` printen op HAL9000: `lpr -PHAL9000 a.cc`)

Gebruik `firefox &` om de web-browser Firefox te starten. De ampersand `&` zorgt er voor Firefox "op de achtergrond" draait, en dat je in de "terminal" kunt doorwerken.

UNIX werkt met processen. CTRL-C stopt een proces, CTRL-Z zet het tijdelijk stop. Verdere controle: `ps`, `top` en `kill`. Voorbeeld, met pipelining: `ps -eaf | grep ikkuh`.

Gebruik "history" via ↑ en ↓, en "tab-completion".

Zie verder dictaat, Hoofdstuk 2.

```
find . -name "*cc" | xargs grep -i jaar
```

```
// Dit is een regel met commentaar ...
#include <iostream> // moet er altijd bij
using namespace std;
const double pie = 3.14159; // een constante (of cmath)
int main ( ) {
  double straal;  // straal van de cirkel
  cout << "Geef straal, daarna Enter .. ";
  cin >> straal;
  if ( straal >= 0 ) {        // accolades nodig!
    cout << "Oppervlakte ";
    cout << pie * straal * straal << endl;
  }//if
  else
    cout << "Niet zo negatief ..." << endl;
  cout << "Einde van dit programma." << endl;
  return 0;
}//main
```

In plaats van

```
cout << pie * straal * straal << endl;
```

is doorgaans

```
double oppervlakte; // van de cirkel
...
oppervlakte = pie * straal * straal;
cout << oppervlakte << endl;
```

beter.

Elk C++-programma bestaat uit:

- speciale symbolen als `+`, `-`, `*`, `/`, `%`, `>=`, `<=`, `<<`, `>>`, `=` (“wordt”), `==` (“is gelijk aan”)
- woordsymbolen als `if`, `while`
- identifiers als `int`, `straal`; vaak zelfgemaakt
- getallen als `42`, `3.14159` (decimale punt)
- strings als `"Einde van dit pRoGrAmMa.  "`
- “whitespace”: spatie (␣), tab, regelovergang (CR/LF)

En dan heb je ook nog: separatoren (␣, `//...`, `/*...*/`), karakters (`'j'`, `'('`, `'\n'`) en literals (getallen, strings).

Bij dit college gebruiken we de volgende C++-keywords:

```
break case char class const delete do
double else enum for if int long new
private public return sizeof static
struct switch this void while
```

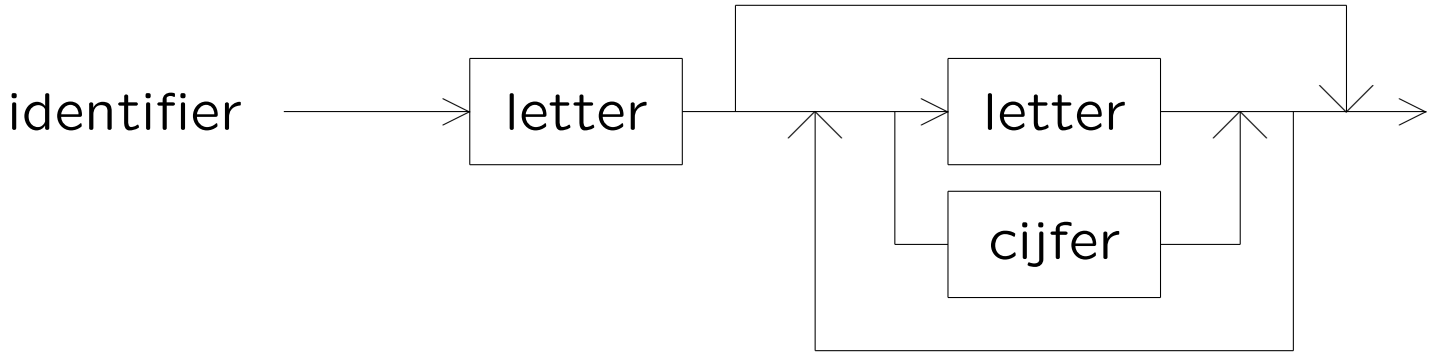
En uit de ANSI-ISO C++-standaard:

```
bool false namespace static_cast true using
```

Er zijn er nog minstens 30 meer. Huidige versie: C++20.

Je kunt alles heel precies grammaticaal vastleggen — en dat moet ook voor de compiler. Daarvoor zijn zogeheten syntax-diagrammen bedacht.

Voorbeeld: een *identifier* is gedefinieerd als *een letter gevolgd door nul of meer cijfers en letters*.



Zie verder: Noam Chomsky, BNF (Backus Naur Form), contekstvrije/formele talen — "loodspet".

Commentaar — 1

Een goed programma bevat veeeeeeeeeeeel commentaar:

```
/*
   Dit is commentaar in C-stijl
   en de compiler slaat het allemaal over!
 */
```

maar niet *te* veel.

Bijvoorbeeld naast "iedere" variabele-declaratie:

```
int stp = 6;  // zes EC studiepunten voor PM
              // als je opgaven EN tentamen haalt
```

Let op met "nesten". Gebruik liever //, en benut /*...*/ bij tijdelijk wegcommentariëren.

Er is verschil tussen // (voor programmeurs, inclusief jezelf) en `cout` (gebruikers, inclusief jezelf: "infoblokje")!

Goed:

```
cout << "Geef eerste voorletter .. ";
cin >> een;
cout << "Geef tweede voorletter .. ";
cin >> twee;
```

Voor de gebruiker onduidelijk:

```
cout << "Geef twee voorletters .. ";
cin >> een >> twee;
```

Overbodig commentaar:

```
cout << "Geef voorletter .. ";
cin >> voorletter;  // lees voorletter in
```

Bovenaan een programma staat ietwat technisch commentaar voor programmeurs: wie, wat, waar(om), wanneer, . . . , welke compiler, . . .

```
//
// Schaakprogramma van A. Turing en M. Carlsen
// C++, 10.9.2021, g++ 3.14159
// ...
...
...
...
  cout << "Welkom ... " << endl;
  cout << "...
  ...
  ...
```

Als een programma begint, krijgt een gebruiker informatie via een "infoblokje" met `cout`-statements: wie, wat, waar, . . . , "wat moet ik doen", . . .

De globale structuur van een C++-programma is:

```
// commentaar: wie, wat, waar(om), wanneer
#include ...
const ...
class ... // allerlei objecten (OOP)
... // nu allerlei functies
int main ( ) {
  ...  // kort
  return 0;
}//main
```

Sommigen zetten de functies *onder* `main` — je hebt dan "prototypes" nodig.

```
int main ( ) {
  int leeftijd; cin >> leeftijd; // foei!
  if ( leeftijd < 12 ) {
    doedit...
  }//if
  else {
    if ( leeftijd <= 67 ) {
      werken...
      return 1; // stopt het programma
    }//if
    doedat...
  }//else
  return 0;
}//main
```

Drie of vier spaties mogen ook . . . mits consequent.

```
int main ( ) {
12int leeftijd; cin >> leeftijd; // foei!
12if ( leeftijd < 12 ) {
1234doedit...
12}//if
12else {
1234if ( leeftijd <= 67 ) {
123456werken...
123456return 1; // stopt het programma
1234}//if
1234doedat...
12}//else
12return 0;
}//main
```

Hierbij staat een 1/2/3/4/5/6 voor een spatie.

C++, een sterk getypeerde taal, kent de volgende basistypes (data-types):

- `int` — geheel getal: `1234`
- `double` — *benadering* van reëel getal: `12.3`
- `bool` — waar (`true`) of niet-waar (`false`)
- `char` — karakter (lettertje): `'3'`

En allerlei zelf-gemaakte types, zie later:

- arrays en strings: `int A[42];` levert array `A` met 42 gehele getallen `A[0]`,`A[1]`,...,`A[41]`, een vector dus
- `class`, `struct`
- pointers (geheugen-adressen)

Gehele getallen — 1

Voor gehele getallen (integers) hebben we het type `int`, voor iets grotere gehele getallen het type `long`.

Als een `int` 4 bytes $= 4 \times 8 = 32$ bits beslaat (gebruik `cout << sizeof (int) << endl;`), is de grootste, `INT_MAX`, gelijk aan (via $2^{10} = 1024 \approx 1000$): $2^{31} - 1 \approx 2 \times 1000^3 = 2 \times 10^9 = 2$ miljard. De kleinste is *ongeveer* `-INT_MAX`.
Soms nodig: `#include <climits>`.

```
01111111  11111111  11111111  11111111
00000000  00000000  00000000  00000001  +
-----------------------------------
10000000  00000000  00000000  00000000
```

Test eventueel `if ( x > INT_MAX - y ) cout << "Te groot";`
en *niet* `if ( x + y > INT_MAX ) cout << "Te groot";`.

↑
zinloze test, altijd `false`

Gehele getallen — 2

```
int getal;          // een geheel getal
int a = 3, b = -5; // en nog twee, nu wel geinitialiseerd

getal = a + b; // getal wordt -2
a = a + 7;     // a wordt met 7 opgehoogd naar 10
      // reken eerst a + 7 uit, en stop die waarde in a
b++;  // hoog b met 1 op (naar -4),
      // hetzelfde als b = b + 1;
--a;  // a wordt 9, hetzelfde als a = a - 1;
getal += a;  // hoog getal met a op (naar 7),
             // hetzelfde als getal = getal + a;
a = 19 / 7;  // a wordt 2: deel 19 door 7
b = 19 % 7;  // b wordt 5: rest bij die deling (modulo)
```

Een `double` (kleinere versie: `float`; typisch 8 en 4 bytes) bevat een benadering van een reëel getal.

Stiekem is een `double` een rationaal getal (uit $\mathbf{Q}$).

Met `#include <cmath>` (vroeger `math.h`) kun je allerlei standaardfuncties als `sqrt`, `sin`, `ceil` (naar boven afronden), `floor` (idem, naar beneden) en `fabs` (absolute waarde) gebruiken.

Zo levert `floor (6.8)` de waarde `6` op, oftewel $\lfloor 6.8 \rfloor = 6$.
En `floor (-6.8)` geeft `-7`, oftewel $\lfloor -6.8 \rfloor = -7$.

(En `#include <ctime>` geeft functies voor de tijd.)

Reële getallen — 2

```
double x;  // een reeel getal
double temp = 36.5, dollar = 0.84152;  // en nog twee
const double pie = 3.28;  // en nog een
int i;     // en een integer

i = 9 / 5;           // dat is 1
x = 9 / 5;           // en weer 1 (1.0000)
x = 9 / 5.0;         // en dat levert 1.8000
x = (double) 9 / 5; // nu 1.8000: (ouderwets) casten
x = static_cast<double>(9) / 5; // idem, (modern) casten
i = 9 / 5.0;  // en dat is weer 1: impliciet casten
i = x + 0.5;  // slim afronden, met impliciet casten
              // misschien beter: (int) ( x + 0.5 )
pie = 3.14;   // verboden, want pie is een constante
```

Hoe druk je `double`'s netjes af?

```
#include <iomanip>
...
double x = 92.36718;
cout << "En x is: "
     << setw (8) // eerstvolgende 8 breed afdrukken
     << setprecision (2) // 2 cijfers na de komma: 92.37
     << setiosflags (ios::fixed|ios::showpoint)
                 // met decimale punt, en 88.00 ipv 88
     << x << endl;
```

Nu kan $14.23 + 18.67 = 32.91$ optreden!
Er zijn allerlei varianten, zie boek . . . (liefhebbers)

O ja, en in plaats van `using namespace std;` is het wellicht beter om altijd `std::cin` en `std::cout` te gebruiken.

Boolese/Booleaanse variabelen, van type `bool`, zijn waar (`true`, 1, in C++: iets wat *ongelijk* 0 is (!!!)) of niet-waar (`false`, 0).
Je kunt zelf ook zo'n type maken via

```
enum boolean { False = 0, True = 1 };
```

Waarheidstabel/tafel:

p	q	!p	p&&q	p||q	p==q
0	0	1	0	0	1
0	1	1	0	1	0
1	0	0	0	1	0
1	1	0	1	1	1
		not	and	or	

Er zijn vele rekenregels, zoals $!\,!\,p = p$ en (de regel van Augustus De Morgan:) $!\,(p \;\&\&\; q) \;= !\,p \,||\, !\,q$.

Enkele voorbeelden van "Boolese expressies":

```
if ( ( 2 < x ) && ( x < 7 ) ) ...
if ( 2 < x < 7 ) ... // NEE! betekent iets anders
if ( y =! 0 ) ...    // FOUT!!! y wordt nu 1 (niet-0)
if ( ! ( ( y < 3 ) || ( y > 7 ) ) ) ... // het-
if ( ( y >= 3 ) && ( y <= 7 ) ) ...     // zelf-
if ( 3 <= y && y <= 7 ) ...             // de
```

En als x gelijk aan 0 is wordt de tweede test niet gedaan:

```
if ( ( x != 0 ) && ( y / x == 7 ) ) ...
```

Dat heet short-circuiting.

```
char letter;                    // een karakter
char let1 = 'q', let2 = '$'; // en nog twee
int i;                          // een integer
i = 'h' - 'c';                  // 5
i = 'a';  // impliciet casten: 97, de ASCII-waarde van 'a'
i = (int) (letter);             // netter (ouderwets)
let1 = let1 + 'A' - 'a';        // bijvoorbeeld 'q' geeft 'Q'
```

Let op het verschil tussen karakters: enkele quotes; en C-strings: dubbele quotes: `"aap"` (`'a' 'a' 'p' '\0'`).

Testen of `letter` de waarde `j` (van "ja") heeft gaat met:

```
if ( letter == 'j' ) ...
```

en dus niet met `if ( letter == j )` of `if ( letter = 'j' )`!

Ieder karakter, een enkele byte, correspondeert met een unieke ASCII-waarde (American Standard Code for Information Interchange) tussen 0 en 255. Gebruik liever geen karakters > 127. En "nooit" expliciet die getallen.

...	36	37	38	...	47	48	49	...	57	58	...
...	$	%	&	...	/	0	1	...	9	:	...

65	66	...	90	91	...	97	98	...	122	123	...
A	B	...	Z	[	...	a	b	...	z	{	...

De CarriageReturn (CR, `'\r'`) heeft ASCII-waarde 13, de LineFeed (LF, `'\n'`) ASCII-waarde 10.
Regelovergangen in de UNIX-wereld: LF, in de Windows-wereld CR en LF.

Sommige karakters gebruiken een escape sequence: `'\n'`.
Ook nog: `'\t'` (tab), `'\\'` (backslash), . . .
Zo geeft `cout << "\\n";` op het beeldscherm `\n`. En bij `cout << "nnn\nnn";` krijg je

```
nnn
nn
```

Een nieuwe regel op het beeldscherm kun je maken met `cout << "\n";` of (beter) met `cout << endl;`. De eerste "buffert" en de tweede "flusht" — en dat is soms fijn.

Met behulp van de toekenning (=, helaas geen := of ←) kun je waarden aan variabelen geven.

```
int getal; bool p; char kar;   getal = 17;
p = false;      // hulpvariabele, betere naam: volwassen
p = ( getal >= 18 ); // het feit of getal minstens 18 is
if ( p ) cout << "volwassen getal ...";
if ( p == true ) ... // mag, niet elegant
if ( p = true ) ...  // HELP, nu wordt p eerst true,
   // is de test altijd waar, en is p ook nog veranderd
if ( getal == 2 || 3 || 5 || 7 || 11 || 13 ) ... // HELP
   // moet zijn: if ( getal == 2 || getal == 3 || getal ...
if ( 'd' <= kar && kar <= 'h' ) ... // is kar d/e/f/g/h?
```

Aan een l-value (laten we zeggen een variabele) mag je een r-value (alles wat een waarde oplevert) toekennen:

```
rij[k] = 42;              // k-de array-element wordt 42
( i < j ? i : j ) = 0; // kleinste van i en j wordt 0!?
```

Met behulp van operatoren (`+ - * / % = << >> ++ --`), variabelen en literals bouw je expressies.

Er gelden zekere prioriteiten. Veel operatoren zijn "links-associatief": `1 + 2 + 3` betekent `( 1 + 2 ) + 3`; daar maakt het trouwens niet uit.
Praktische aanpak: zet overvloedig haakjes: `(` en `)`.

Omdat `=` "rechts-associatief" is mag je statements schrijven zoals `x = y = 51;`, en dat betekent `x = ( y = 51 );`. Nu worden `x` en `y` beide `51`: de "waarde van een toekenning" is "dat wat je toekent".

Bij `f(x) + g(x)` is het in C++ onbepaald of eerst `f(x)` of eerst `g(x)` berekend wordt.

- maak de eerste programmeeropgave — de deadline is op maandag 27 september 2021!
- verslag (in LaTeX) $\longrightarrow$ volgende week
- lees Savitch Hoofdstuk 1 (nog eens), begin met 2
- lees dic-taat Hoofdstuk 1, 2 (nog eens), begin met 3
- maak opgaven 1/5 uit het "opgavendictaat"
- www.liacs.leidenuniv.nl/~kosterswa/pm/

Programmeermethoden

Controle-structuren

Walter Kosters en Jonathan Vis

week 3: 20–24 september 2021

www.liacs.leidenuniv.nl/~kosterswa/pm/

maandag 20 september	di	wo	donderdag 23 september	vr
16:15–. . . Snellius 302,306 “informeel” vragenuur			11:15–13:00 Gorlaeus zaal 1 $\leqslant 75$ college 3 “iedereen” A-G	
			14:15–. . . Snellius 302,306,. . . ,313 werkcollege 3, vragenuur Informatica	

maandag 27 september	di	wo	donderdag 30 september	vrijdag 1 oktober
14:15–16:00 Snellius 303,312,313,174 werkcollege 3 Wiskunde neem laptop mee			11:15–13:00 Gorlaeus zaal 1 college 4 iedereen!	14:15–16:00 Snellius 302,306,313 werkcollege 4 Wiskunde
14:15-. . . Snellius 302,306 vragenuur iedereen 17:00 deadline			14:15–. . . Snellius 302,306,. . . ,313 werkcollege 4, vragenuur Informatica	16:15–. . . Snellius 412,407 vragenuur Wiskunde

Eerste programmeeropgave

Programmeermethoden 2021
Eerste programmeeropgave: abc-formule

De *eerste* programmeeropgave van het vak Programmeermethoden in het najaar van 2021 heet *abc-formule*; zie ook het eerste, tweede en derde werkcollege.

Deze opgave probeert te bepalen of iemand geschikt is voor een studie aan de universiteit: er is immers geen loting. Daartoe moeten enkele vragen beantwoord worden; zo moet de kandidaat weten op welke dag hij/zij geboren is. En als je de abc-formule niet kunt gebruiken, is een beta-studie misschien niet verstandig.

Om te beginnen moet de gebruiker zijn/haar geboortejaar als geheel getal invoeren, en daarna de geboortemaand, ook als geheel getal. Vervolgens voert hij/zij de geboortedag in, wederom als geheel getal. Het programma berekent dan de leeftijd van de gebruiker, zowel in aantal jaren als in maanden (bijvoorbeeld: 10 jaar en 3 maanden; 123 maanden); beide worden op het beeldscherm getoond. De leeftijd in maanden wordt analoog aan die in jaren bepaald (als je op de 31ste geboren bent, wordt je iedere maand een maand ouder, maar je bent niet zo vaak "maandig" — dat ben je namelijk alleen op iedere 31ste). Jarige en maandige gebruikers worden gefeliciteerd. Aangenomen mag worden dat het programma op de peildatum *27 september 2021* draait (gebruik `const`; liefhebbers mogen met `ctime` de echte huidige dag opvragen en gebruiken). Let op: het programma moet in principe ook op andere peildata vanaf heden tot 2100 correct werken!
Gebruikers jonger dan 10 jaar (de 10-de verjaardag nog niet gevierd) of ouder dan 100 jaar (dus 101-ste verjaardag reeds gevierd) worden meteen geweigerd. Als uit het geboortejaar direct al duidelijk is dat het met de leeftijd niets gaat worden, hoeven de vragen naar maand en/of dag niet gesteld te worden. Maar soms biedt pas de dag uitsluitsel!
Nu moet de gebruiker zijn/haar geboortedag (zondag, maandag, ..., zaterdag) weten. Als deze fout is, wordt men meteen "verwijderd", en stopt het programma. Het antwoord moet met *één letter* (de eerste letter van de dag; geen cijfer dus) worden gegeven, bijvoorbeeld w voor woensdag. In het geval van d/z wordt nog om de tweede letter gevraagd.
Het is *niet* de bedoeling `ctime` te gebruiken om deze dag uit te rekenen. Het programma moet een zelf bedachte berekening bevatten om deze dag te bepalen! Gebruik bijvoorbeeld dat 1 januari 1901 op een dinsdag viel. Gebruik *niet* het Doomsday algoritme (zie ook hier), en ook *niet* allerlei ingewikkelde formules. Voor de periode 1901–2099 geldt dat een jaar een schrikkeljaar is precies dan als het jaartal door 4 deelbaar is.

De echte test bestaat uit enkele vragen. Mensen van 30 jaar of ouder worden hierbij twee maal "netter" aangesproken dan jongeren. Splits de C++-code in het programma niet onnodig vaak!
Er wordt gekeken of de aanstaande student de abc-formule kan gebruiken. Wiskundig inzicht is namelijk vereist voor een beta-studie. Mocht dat niet zo zijn, wordt er getest hoe het met de kunst- of literatuurkennis staat.

Een kwadratische vergelijking heeft 0, 1 of 2 reële oplossingen, die we kunnen vinden met behulp van de abc-formule. Het programma genereert een willekeurige kwadratische vergelijking van de vorm $a\,x^2 + b\,x + c = 0$ en toont deze "netjes" op het scherm (de exponent mag er uit zien als `x^2`). Hierbij geldt dat *a*, *b* en *c* gehele getallen zijn (in absolute waarde maximaal 1000000), waarbij *a* groter dan 0 is. De student wordt gevraagd hoeveel reële oplossingen zij/hij denkt dat deze vergelijking heeft: 0, 1 of 2. In alle gevallen worden de oplossingen van de vergelijking (als die er zijn) op het scherm afgedrukt. Liefhebbers mogen met complexe getallen werken.
Voor *a*, *b* en *c* moeten `int`'s gebruikt worden. Omdat de "discriminant" te groot kan zijn voor een `int`, moet een `double` gebruikt worden in de berekeningen. Voor worteltrekken kan `y = sqrt (x)` worden gebruikt; sqrt zit in `cmath`. Voor het fabriceren van willekeurige gehele getallen moet gebruik worden gemaakt van de random-generator uit C++. Gebruik bijvoorbeeld `x = rand ( ) % 20;` om een "willekeurig" getal tussen 0 en 19 (grenzen inbegrepen) in de `int` variabele `x` te krijgen. Zet bovenaan in main: `srand (42);` of `srand (jaar);` (nadat `jaar` een waarde heeft gekregen), om de random-generator eenmalig te initialiseren. In plaats van 42 mag ook een ander getal staan — of zelfs, voor liefhebbers, de tijd. En soms is hiervoor `#include <cstdlib>` nodig, helemaal bovenaan het programma.

Is het antwoord goed, dan wordt de kandidaat tot een exacte studie toegelaten, en stopt het programma. Anders wordt één meerkeuzevraag (Aa/Bb/Cc/Dd) over kunst of literatuur gesteld, die uitsluitsel biedt over de toelating tot een alpha-studie. Als het daar ook mis gaat, is men helaas niet geschikt voor een universitaire studie. Gebruikers tot of tot en met (kies zelf) 30 jaar krijgen hier een andere vraag dan de oudere gebruikers

— maar bij beiden is "hetzelfde" antwoord, bijvoorbeeld steeds B, goed. Of het antwoord goed of fout is, het juiste antwoord wordt steeds op het scherm afgedrukt.

Opmerkingen

Als de gebruiker een niet bestaande maand invoert, bijvoorbeeld −8, of een jaartal als 4242 (in de toekomst dus), stopt het programma met de mededeling dat dit niet kan (gebruik `return 1;`). Evenzo voor een niet bestaande dag, bijvoorbeeld 31 april of 42 december. We nemen aan dat de gebruiker zo vriendelijk is verder geen fouten te maken bij het invoeren van gegevens: hij/zij voert niet al te gekke getallen of letters in, etcetera. Vanzelfsprekend worden hem/haar wel duidelijke vragen gepresenteerd.

Elk programma moet bij het "runnen" aan het begin op het beeldscherm laten zien wie de makers zijn, wat hun jaar van aankomst, studierichting en studentnummer is, welke opgave het is, wat de gebruiker te wachten en te doen staat, de datum waarop het programma gemaakt is, enzovoorts. Dit noemen we het infoblokje.
Probeer dit er netjes uit te laten zien. Maak geen al te complexe kaders eromheen; gebruik liefst alleen de eerste 128 gewone karakters.
Bovenaan het programma (in de C++-code dus) staat uiteraard commentaar, waarin een aantal van deze elementen ook weer terugkomen, maar dan meer gericht op programmeurs, bijvoorbeeld de naam van de gebruikte compiler.

Denk aan het gebruik van lege regels, inspringen, commentaar, constanten, enzovoorts. Bovenaan het programma dient zoals gezegd commentaar over het programma te staan, speciaal bestemd voor andere *programmeurs* (en nakijkers), bijvoorbeeld kort wat het programma doet, en welke compiler gebruikt is: gebruikers van het programma vinden dat laatste niet interessant. Het infoblokje moet tijdens het "runnen" van het programma op het scherm komen, en is bestemd voor *gebruikers* van het programma. Lees ook eens over richtlijnen bij het maken van programmeeropgaven, en bestudeer de huisregels. Er hoeft geen gebruik van functies, arrays en het while- en for-statement gemaakt te worden. Alleen de headerfiles `iostream` mag en moet gebruikt worden — en eventueel `ctime` voor liefhebbers; en misschien `cstdlib` voor het gebruik van de random-generator. Ruwe indicatie voor de lengte van het C++-programma: 200 regels (300 mag ook wel).
Letters moeten als `char` worden ingelezen, dus *niet* met strings, die mogen namelijk niet gebruikt worden.

Uiterste inleverdatum: maandag 27 september 2021, 17:00 uur.
De manier van inleveren (één exemplaar per koppel, dat — ter herinnering — uit maximaal twee personen bestaat) is als volgt. Digitaal de C++-code inleveren: stuur een email naar `pm@liacs.leidenuniv.nl`; en een print van het verslag in de doos bij kamer 159 van het Snellius. Stuur geen executable's, LaTeX-files of PDF-files, lever alleen één C++-file digitaal in! Noem deze bij voorkeur `bidenharris1.cc`, dit voor de eerste opdracht van het duo Harris / Biden. De laatst voor de deadline ingeleverde versie wordt nagekeken.
Tip: maak een nette e-mail, met een korte maar zinnige tekst als inhoud, en de C++-file als attachment. Overal duidelijk datum en namen van de twee makers vermelden, in het bijzonder als commentaar in de eerste regels van de C++-code. Lees bij het derde werkcollege hoe het verslag eruit moet zien. Zijn spaties/tabs goed verwerkt?

Te gebruiken compiler: als hij maar C++ vertaalt; het programma moet in principe zowel op een Linux-machine (met `g++`) als onder Windows met Code::Blocks draaien. Test dus zo mogelijk op beide systemen! Normering: (consequente) layout 2; commentaar 2; infoblokje 1; verslag 1; werking 4. Eventuele aanvullingen en verbeteringen: lees de WWW-bladzijde die je nu ziet: `www.liacs.leidenuniv.nl/~kosterswa/pm/op1pm.php`.
Recente aanvullingen en/of wijzigingen staan in rood.

www.liacs.leidenuniv.nl/~kosterswa/pm/op1pm.php

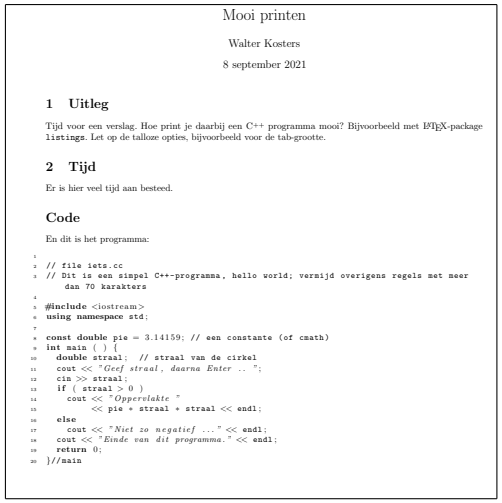
Inleveren: digitaal `bidenharris1.cc` als attachment per e-mail sturen naar `pm@liacs.leidenuniv.nl`; en `mooi.pdf` printen en in de doos bij Snellius 159 doen.

```
\usepackage{listings}
\begin{document}
% ... etcetera ...
```

`mooi.tex`

```
// Opgave 1: ...
#include <iostream>
using namespace std;
// ... etcetera ...
```

`bidenharris1.cc`



Mooi printen

Walter Kosters

8 september 2021

1 Uitleg

Tijd voor een verslag. Hoe print je daarbij een C++ programma mooi? Bijvoorbeeld met LaTeX-package `listings`. Let op de talloze opties, bijvoorbeeld voor de tab-grootte.

2 Tijd

Er is hier veel tijd aan besteed.

Code

En dit is het programma:

```
// file iets.cc
// Dit is een simpel C++-programma, hello world; vermijd overigens regels met meer
    dan 70 karakters

#include <iostream>
using namespace std;

const double pie = 3.14159; // een constante (of cmath)
int main ( ) {
  double straal;  // straal van de cirkel
  cout << "Geef straal, daarna Enter .. ";
  cin >> straal;
  if ( straal > 0 )
    cout << "Oppervlakte "
         << pie * straal * straal << endl;
  else
    cout << "Niet zo negatief ..." << endl;
  cout << "Einde van dit programma." << endl;
  return 0;
}//main
```

1

`mooi.pdf`

Zie de uitleg bij het derde werkcollege.

We werken met Donald Knuth's LaTeX, zie de video. Zorg ervoor dat de files `mooi.tex` en `bidenharris1.cc` in één map = directory staan, en tik in een Linux-terminal in: `pdflatex mooi.tex` (installeer `texlive-full`). Ook op Windows verkrijgbaar. Dit produceert `mooi.pdf`.

Mooie formules: `$x^2 + y_{42} = z$` levert $x^2 + y_{42} = z$.

Er zijn allerlei tools (Lyx, Kile, TeXworks, . . .) om dit te ondersteunen.

LaTeX-video

En op internet: Overleaf, waar je ook gratis met twee personen kunt samenwerken ("Share").

C++ kent de volgende controle-structuren:

keuze `if` (met als variant: `switch`)

onbekend (maar eindig?) aantal herhalingen

`while` (met als variant `do ... while`)

“vast” aantal herhalingen `for`

We gebruiken geen labels/goto’s!

Een if-statement gaat informeel als volgt:

```
als ( een of andere test )
  als de test waar is: zus en zo
anders
  als de test onwaar is: dit en dat
```

En een while-statement gaat informeel als volgt:

```
zolang ( een of andere test waar is )
  zus en zo
```


```
if ( temperatuur > 0 )
  fietsen (...);                    ← als test ≠ 0
else {
  paraplukopen (...);               ← als test = 0
  lopen (...);
}//else
```

Let op de accolades om het "compound" (samengestelde) statement. Rond `fietsen (...);` *mogen* accolades.

De tests gebruiken predicaten als `==` (*gelijk aan?*), `!=` (*ongelijk aan?*), `<` (*kleiner dan?*), `>` (*groter dan?*), `<=` (*kleiner dan of gelijk aan?*) en `>=` (*groter dan of gelijk aan?*).

Waar hoort een “else” bij?

```
if ( x > 0 )
  if ( y > 0 )
    cout << "Beide groter dan nul.";
  else                          // waar hoort deze bij?
    cout << "                 ? ? ? ?             ";
```

Overigens: met accolades vermijd je veel problemen!

Waar hoort een “else” bij?

```
if ( x > 0 )
  if ( y > 0 )
    cout << "Beide groter dan nul.";
  else                              // waar hoort deze bij?
    cout << " x positief, y negatief (of 0)  ";
```

Bij de laatste nog “openstaande” if!

Zorg ervoor dat de layout klopt — de compiler kijkt daar niet naar. En: “kloppen = mooi/consequent/simpel†/...”

† vergelijk Occam’s razor

```
if ( x == 0 ) x = 1;
else if ( x == 2 ) x = 7;
```

OK

```
if ( x == 0 )
  x = 1;
else if ( x == 2 )
  x = 7;
```

OK++

```
if ( x == 0 )
  x = 1;
else
  if ( x == 2 )
    x = 7;
```

OK

```
if ( x == 0 ) x = 1;
else if ( x == 2 ) x = 7;
```

OK (zie ook switch)

```
if ( x == 0 ) x = 42;
if ( x == 2 ) x = 7;
```

```
// wat als hier x = 2;?
```

zwak (x wordt twee maal lastig gevallen als x toevallig 0 is)

```
if ( x == 0 ) {
  cout << "...iets..." << endl;
  return 1;
}//if
if ( x == 2 ) x = 7;
```

```
// geen else nodig
```

OK++

```
if ( code == 0 )  // code van type int
  doedit (...);
else if ( code == 1 )
  doedat (...);
else ...
```

is equivalent met:

```
switch ( code ) {
  case 0: doedit (...); break;  // layout ??
  case 1: doedat (...); break;
  ...
}//switch
```

Je kunt ook als geval `default:` benutten.
Typisch gebruik: menu-opties, met een `char`.

Stel je wilt de eerste `n` positieve gehele getallen en hun kwadraten afdrukken:

```
int i, n;  cin >> n;                                  1--1
                                                      2--4
i = 1;                                                3--9
while ( i <= n ) {                                    4--16
  cout << i << "--" << i * i << endl;                 5--25
  i++;                                                6--36
}//while                                             ...
```

Het kan ook zo:

```
for ( i = 1; i <= n; i++ )
  cout << i << "--" << i * i << endl;
```

Nog enkele voorbeelden van while-loops:

```
while ( n != 0 )  // zolang n niet 0 is; hopelijk n >= 0
  n--;            // laag hem met 1 af: n = n - 1
```

Dit is overigens hetzelfde als: `while ( n ) n--;`

Let er op dat je beter `( n > 0 )` als test kunt gebruiken:

```
x = 1;
while ( x < 100 ) { cout << x << endl; x = x + 2; }
```

drukt de oneven getallen < 100 af, maar niet als de test `( x != 100 )` zou luiden — dan stopt het niet.

Bij een while-loop is het aantal "doorgangen" van te voren vaak onbekend of lastig te bepalen:

```
int x = 1;
while ( x < 1000 ) x = 2 * x;
// nu is x gelijk aan 1024
```

Het Collatz-probleem ($3x + 1$ vermoeden) zegt dat het volgende programma voor ieder positieve gehele `x` stopt:

```
while ( x != 1 )  // 13->40->20->10->5->16->8->4->2->1
  if ( x % 2 == 0 )  // is x even?
    x = x / 2;
  else
    x = 3 * x + 1;
```

Als de loop stopt, is `x` na afloop gelijk aan 1.

Na

```
while ( x % 2 == 0 ) x = x / 2;       // of x /= 2;
```

geldt `! ( x % 2 == 0 )`, oftewel `x` is nu oneven geworden: alle factoren 2 zijn uit de "oude" `x` gehaald.

Hier staat in feite: zolang `x` even is, deel `x` door 2, dus bijvoorbeeld $56 \rightarrow 28 \rightarrow 14 \rightarrow 7$.

Als een while-loop stopt is na afloop de test "onwaar".

Er mogen accolades `{` en `}` rond `x = x / 2;` staan.

Bij een for-loop als

```
for ( i = 3; i <= 17; i = i + 2 ) cout << i << "-";
```

wordt eerst één maal de initialisatie `i = 3;` gedaan, en daarna herhaald de cyclus bestaande uit test (`i <= 17`), "body" (hier statement `cout << i << "-";`) en aanpassing van de teller (`i = i + 2`).

We krijgen uitvoer `3-5-7-9-11-13-15-17-`, en na afloop heeft `i` de waarde `19`.

Opgave: probeer het streepje na `17` kwijt te raken.

Een for-loop is eigenlijk hetzelfde als een while-loop: “for (A B C) D” komt overeen met “A while (B) { D C }”. Bij een for-loop is het aantal “doorgangen” vaak bekend.

Een rare for-loop:

```
for ( ; ! ( x % 2 ); x /= 2 ) ;
```

Dit is hetzelfde als

```
while ( ! ( x % 2 ) ) x = x / 2;
```

En wat te denken van `for ( ; ; ) ;`? Dat is een oneindige loop, met een “empty statement”, net als `while ( true ) ;`.

Een dubbele for-loop:

```
int i, j;
for ( i = 1; i <= 5; i++ ) { // buitenste loop
  cout << i << ": ";
  for ( j = 1; j <= i; j++ ) // binnenste loop
    cout << i * j << " ";
  cout << endl;
}//for
```

geeft:

```
1: 1
2: 2 4
3: 3 6 9
4: 4 8 12 16
5: 5 10 15 20 25
```

Het do-while statement is een variant op de while-loop, waarbij de "body" in ieder geval één maal wordt uitgevoerd:

```
do {
  cout << "Geef positief getal .. ";
  cin >> getal;
} while ( getal < 0 );

do
  cin >> kar;
while ( ( 'a' <= kar && kar <= 'z' )
        || ( 'A' <= kar && kar <= 'Z' ) );
// nu bevat kar de eerste "niet-letter"
```

NB Pas op met `cin >> ...`, dit slaat whitespace over.

- maak de eerste programmeeropgave — de deadline is op maandag 27 september 2021, 17:00 uur

- lees de tweede programmeeropgave:
  www.liacs.leidenuniv.nl/~kosterswa/pm/op2pm.php

- lees Savitch Hoofdstuk 2

- lees dictaat Hoofdstuk 3, tot en met 3.5, en 3.7

- maak opgaven 6/10 uit het opgavendictaat

Programmeermethoden

Functies & files

Walter Kosters en Jonathan Vis

week 4: 27 september–1 oktober 2021

www.liacs.leidenuniv.nl/~kosterswa/pm/

maandag 27 september	di	wo	donderdag 30 september	vrijdag 1 oktober
14:15–16:00 Snellius 303,312,313,174 werkcollege 3 Wiskunde neem laptop mee			11:15–13:00 Gorlaeus zaal 1 college 4 iedereen!	14:15–16:00 Snellius 302,306,313 werkcollege 4 Wiskunde
14:15-. . . Snellius 302,306 vragenuur iedereen 17:00 deadline			14:15–. . . Snellius 302,306,. . . ,313 werkcollege 4, vragenuur Informatica	16:15–. . . Snellius 412,407 vragenuur Wiskunde

Vanaf volgende week:

- maandag 16:15–. . . : “informeel” vragenuur in 302/306
- donderdag 11:15–13:00: college in Gorlaeus zaal 1/2/3
- donderdag 14:15–16:00: werkcollege in 302/. . . /313
- donderdag 16:15–. . . : vragenuur in 302/306
- vrijdag 14:15–16:00: werkcollege in 302/306/313
- vrijdag 16:15–. . . : vragenuur in 412/407

Een functie is een zwarte doos (black box) waar informatie in gaat en informatie uit komt.

Elk C++-programma bestaat uit een stel functies, onder elkaar. Executie begint bij de functie `main`.

Sommige functies rekenen iets uit (zoals `int`-functies: geef het kwadraat van x terug), andere verrichten een taak (`void`-functies: druk een tabel af op het scherm, of afwassen; geef `void` = leeg = niets terug, doe alleen wat).

Functies hebben allerlei (soorten) parameters, die ze ook kunnen aanpassen.

Functies mogen in C++ alleen functies aanroepen die eerder gedefinieerd zijn.

Een eenvoudige `void`-functie:

```
void tekstOpScherm ( ) { // heel kort infoblokje
  cout << "Sterke tekst." << endl;
}//tekstOpScherm
```

En een eenvoudige `int`-functie:

```
int inhoud (int lengte, int breedte, int hoogte) {
  return lengte * breedte * hoogte;
}//inhoud
```

Met aanroepen:

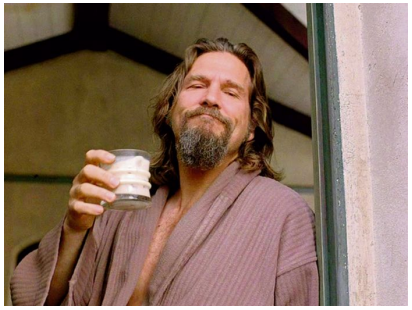
```
tekstOpScherm ( );
cout << inhoud (16,37,42) << endl;
```

Verwissel = swap

Een functie die zijn parameters omwisselt:

```
void wissel (int & x, int & y) {
  int hulp = x;
  x = y;
  y = hulp;
}//wissel
```

call by reference: &



Met aanroep:

```
int a = 33, x = 88;
cout << a << " en " << x << endl; // 33 en 88
wissel (a,x);
cout << a << " en " << x << endl; // 88 en 33
```

Hoe werkt het functie-mechanisme?

Bij aanroep "spring" je naar de desbetreffende functie, en als die klaar is, wanneer je een `return` of de laatste `}` tegenkomt, "spring" je weer terug, en wel naar het "return-adres". Parameters worden netjes doorgegeven.

Soms helpt het als je niet aan het "springen" denkt, maar meer denkt in termen van "deze functie verricht die taak".

Eigenlijk komen functie-aanroepen op een stapel gevuld met "uitgestelde verplichtingen".

```
// bereken inhoud van lengte bij breedte bij hoogte blok
double inhoud (double lengte, double breedte,
               double hoogte) {
  double temp;
  temp = lengte * breedte * hoogte;
  return temp;
}//inhoud
```

Hier zijn `lengte`, `breedte`, `hoogte` en `temp` locale variabelen, waarbij `lengte`, `breedte` en `hoogte` (de formele parameters) als startwaarde de waarde van de actuele parameters krijgen; ze worden *wel* geinitialiseerd, in tegenstelling tot `temp`. Hun scope — waar ze leven — is de functie `inhoud`. Men noemt `lengte`, `breedte` en `hoogte` wel call by value parameters. Bij een aanroep als `t = inhoud (breedte,5,x);` zijn `breedte`, `5` en `x` de actuele parameters (of variabelen).

De volgende functie bepaalt of `jaar` een schrikkeljaar is:

```
// is jaar een schrikkeljaar?
bool schrikkel (int jaar) {
   return ( jaar % 4 == 0
            && ( jaar % 400 == 0 || jaar % 100 != 0 ) );
}//schrikkel
```

Dus 1963 niet, 2018 niet, 2020 wel, 2000 wel, en 2100 niet . . .

De grootste gemeenschappelijke/gemene deler (ggd) van twee positieve gehele getallen ($\geqslant 0$, niet beide 0) wordt met het algoritme van Euclides als volgt berekend:

```
int ggd (int x, int y) {
  int rest;
  while ( y != 0 ) {
    rest = x % y;  x = y;  y = rest;
  }//while
  return x;
}//ggd
```

Voorbeeldaanroepen:

```
  cout << ggd (15,21) << endl;
  z = ggd (z,7);  // z van type int
```

Een functie kan maar één waarde retourneren = teruggeven. (Of zelfs geen, bij een `void`-functie.)

Hoe kun je dan twee of meer waarden genereren?

Antwoord: met "call by reference", let op de &.

Overigens: een `void`-functie hoeft geen `return`-statement te hebben, maar het mag wel. Er staat dan *geen* waarde achter, dus gewoon `return;` stopt zo'n functie.

```
// vereenvoudig breuk teller/noemer zoveel mogelijk
// aanname: teller >= 0, noemer > 0
void vereenvoudig (int & teller, int & noemer) {
  int deler = ggd (teller,noemer);
  if ( deler > 1 ) { // test hoeft niet
    teller = teller / deler;
    noemer = noemer / deler;
  }//if
}//vereenvoudig
```

Voorbeeldaanroep:

```
  int tel = 15, noem = 21;
  vereenvoudig (tel,noem);
  cout << tel << " " << noem << endl;
```

Boven iedere functie hoort duidelijk commentaar:

```
// vereenvoudig breuk teller/noemer zoveel mogelijk
// aanname: teller >= 0, noemer > 0
void vereenvoudig (int & teller, int & noemer) {
  ...
}//vereenvoudig
```

Tip: maak een zin waarin de functienaam en de namen van de parameters voorkomen.
En: wat geldt vooraf, en wat na afloop?

Naast call by value, waar de *waarde* van de variabele aan een "lokale kopie" wordt doorgegeven, bestaat ook call by reference, waar de variabele zelf, of preciezer: diens *adres*, wordt doorgegeven.

```
// wissel inhoud van a en b
void wissel (int & a, int & b) {
  int hulp = a;
  a = b;
  b = hulp;
}//wissel
```

Voorbeeldaanroep: `a = 8; k = 2; wissel (a,k);`

De & (ampersand) geeft aan dat het een call by reference variabele betreft.

Voorbeeld — call by value

```
void hoogop (int x) { x = x + 10; cout << x; }//hoogop

void maaknul (int t) { t = 0; cout << t; }//maaknul
```

```
int x, m, q;
x = 7; hoogop (x); cout << x;          17 7
m = 3; hoogop (m+8); cout << m;        21 3
q = 5; maaknul (q); cout << q;         0 5
maaknul (42);                          0
```

Er wordt alleen een *waarde* doorgegeven, en wel van de *actuele* parameter aan de *formele* parameter; er wordt dus een "lokale kopie" gemaakt, wat tijd en ruimte kost.

Voorbeeld — call by reference

```
void hoogop (int & x) { x = x + 10; cout << x; }//hoogop

void maaknul (int & y) { y = 0; cout << y; }//maaknul

  int x, m, q;
  x = 7; hoogop (x); cout << x;          17 17
  m = 3; hoogop (m+8); // VERBODEN!!!
  q = 5; maaknul (q); cout << q;         0 0
  maaknul (42);  // VERBODEN!!!
```

Er wordt nu een *adres* (een *pointer*) doorgegeven. De *actuele* parameter kan nu wel veranderen. De *actuele* parameter mag geen "rare" expressie als `m+8` of `42` zijn. Er wordt alleen een *adres* gekopieerd.

En dan nu: files.

Input en output voor programma's staan vaak in files, bijvoorbeeld `iets.cc`, `uitvoer.txt`, `cin` (toetsenbord) en `cout` (beeldscherm).

Voor de tweede programmeeropgave moet je een C++-programma schrijven dat een file keurig afdrukt, daarbij //-commentaar verwijdert, optredens van een gegeven drieletterwoord telt en het Collatz-vermoeden controleert voor zekere getallen uit de file.

```
#include <fstream>
...
ifstream invoer ("jefile.txt", ios::in);
ofstream uitvoer ("./C++/iets.cc", ios::out);
char letter;              // zelfs / bij Windows!
...
letter = invoer.get ( );
uitvoer.put (letter);
uitvoer << "Hitchcock";
...
invoer.close ( );
uitvoer.close ( );     // niet vergeten!
```

Hier is `invoer` de *variabele* die de file voorstelt, die in het echt `jefile.txt` heet.

Een file is een "object" van "klasse" `ios`. Ook `cin` en `cout` zijn van deze klasse. Met objecten kun je bepaalde dingen doen: "memberfuncties" (= "methoden") aanroepen, zoals `get`. Je zegt dan de naam van het object, dan een punt, en dan de naam van de methode.

Voorbeelden:

```
letter = invoer.get ( );
cout.put (letter);
```

Eigenlijk is een invoerfile van klasse (= type) `ifstream`, en een uitvoerfile van klasse `ofstream`. Beide stammen af van `ios`. En `get` en `put` zijn (member)functies.

Een tekstfile, zoals een C++-programma, bestaat uit regels, gescheiden door regelovergangen (bij UNIX LF, bij Windows CR-LF). Meestal staat aan het eind ook een regelovergang, soms gevolgd door het "einde-file (EOF) symbool". Daarop kun je testen met de methode `eof ( )`.

Zo kopiëren we een file `invoer` naar een file `uitvoer`:

```
kar = invoer.get ( );   // eerst een maal get-ten!!!
while ( ! invoer.eof ( ) ) {
  uitvoer.put (kar);
  kar = invoer.get ( ); // en hier alle volgende ...
}//while
```

Het lijkt alsof er één `get` meer wordt gedaan dan `put`'s, maar de `close` zet als het ware de als laatste gelezen EOF. (Eigenlijk is dit het UNIX-commando `cp`.)

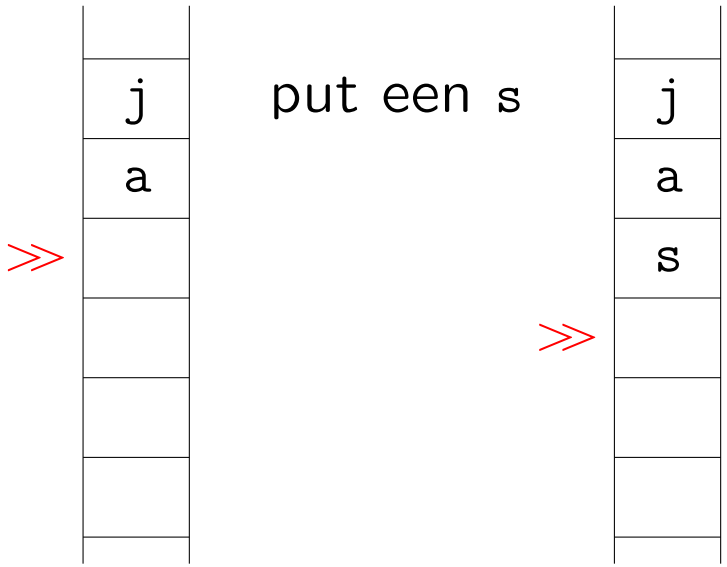
Wijzig stap voor stap zo'n kopieerprogramma:

```
kar = invoer.get ( );
while ( ! invoer.eof ( ) ) {
                              // wijzig dit
  if ( kar != '\n' )          // stap voor stap
    uitvoer.put (kar);        // voor de tweede
                              // programmeeropgave
  kar = invoer.get ( );
}//while
```

Dit kopieert, maar sloopt alle regelovergangen weg.

Meer `get`'s zijn niet nodig!

Eigenlijk werken files met filepointers, net als bij oude video-banden. Voorlopig kun je alleen vooruit spoelen.
Een `put` zet een karakter neer en schuift de filepointer één op, `get` pakt een karakter en schuift de filepointer ook op.



Iets algemener:

```
string filenaam;  // gebruik <string>
ifstream invoer;  // gebruik <fstream>
...
cin >> filenaam;
invoer.open (filenaam.c_str ( )); // in C++11 hoeft
if ( invoer.fail ( ) ) {          // ".c_str ( )" niet
  cout << filenaam << " niet te openen" << endl;
  return 1;  // of  exit (1);  of ...
}//if
```

PS En files doorgeven als parameter:

```
void doewat (ifstream & invoer, ofstream & uitvoer) ...
```

Voor de tweede programmeeropgave moet je een C++-programma schrijven dat een gegeven file netjes ingesprongen afdrukt, en daarbij //-commentaar verwijdert:

```
␣␣if␣(␣leeftijd␣␣>␣6171/97␣)␣␣{{␣//␣commentaar!
cout␣<<␣"Oud";//␣64␣of␣niet?

if␣(␣leeftijd␣␣>␣6171/97␣)␣␣{{␣
␣␣␣␣␣␣cout␣<<␣"Oud";
```

Hier is ␣ een spatie. En "tab = 3".

Hoe vaak komt `mag` voor? Klopt Collatz voor 6171?

www.liacs.leidenuniv.nl/~kosterswa/pm/op2pm.php

Tweede programmeeropgave

Programmeermethoden 2021
Tweede programmeeropgave: Netjes

De *tweede* programmeeropgave van het vak Programmeermethoden in het najaar van 2021 heet *Netjes*; zie ook het vierde werkcollege, vijfde werkcollege (de betreffende WWW-bladzijde bevat handige tips) en zesde werkcollege, en lees geregeld deze pagina op WWW.

Er moet een programma worden geschreven dat een foutloos te compileren C++-programma (bijvoorbeeld een uitwerking van de eerste programmeeropgave) een klein beetje probeert te begrijpen, oftewel een paar zaken doet die een compiler ook moet doen. Het programma moet de invoerfile netjes ingesprongen naar een uitvoerfile kopiëren, en daarbij alle `//`-commentaar weglaten. En tellen hoe vaak een gegeven drietal letters direct achter elkaar voorkomt. Verder moet het programma getallen opsporen, en kijken of deze een speciale eigenschap hebben.
Stel allereerst enkele eenvoudige vragen om gegevens van de gebruiker te weten te komen. Gevraagd wordt hoe de originele invoerfile en de "doelfile" heten (als de invoerfile niet bestaat stopt het programma direct), en hoe groot de parameter `tab` (zie straks) moet worden. Het programma leest dan eenmalig de opgegeven invoerfile, en schrijft deze symbool voor symbool op de juiste wijze aangepast weg naar de uitvoerfile; na afloop wordt een rapportje op het scherm afgedrukt.

De volgende vier punten moeten worden geadresseerd:

1. Commentaar moet verwijderd worden.
   Elk commentaar dat begint met `//` moet niet naar de uitvoerfile worden weggeschreven, maar weggelaten worden. Alleen de regelovergang wordt weer naar de uitvoerfile gekopieerd.
   Voor het gemak mag aangenomen worden dat er geen `/* ... */` commentaar voorkomt. Als er binnen `//`-commentaar opnieuw `//` voorkomt, wordt die volgende `//` ook gewoon verwijderd. We vatten zelfs `//` binnen een string op als het begin van commentaar.
2. Inspringen moet netjes geregeld worden.
   De bedoeling is dat iedere regel op *diepte* `d` even ver inspringt, en wel `tab` maal `d` spaties. Hierbij is `tab` een door de gebruiker te kiezen getal, bijvoorbeeld 3. De diepte `d` van een regel wordt als volgt bepaald. De eerste regel van het programma is op diepte 0, iedere openingsaccolade `{` verhoogt de diepte met 1 (één) en iedere sluitaccolade `}` verlaagt de diepte met 1 (één). Een veranderde diepte merk je pas op de volgende regel (zie verderop hoe we accolades zelf afhandelen).
   We nemen aan dat er geen accolades binnen strings (of als karakter `'{'`) voorkomen. Accolades die binnen commentaar staan tellen niet mee voor de berekening van de diepte. De oude regelstructuur van het programma blijft behouden; voor iedere regel geldt dat het eerste karakter ongelijk spatie en TAB (`'\t'`) op positie `tab` maal `d` +1 moet komen. Dit karakter kan een regelovergang zijn; zo wordt een oorspronkelijk "lege" regel op diepte `d` nu een regel met `tab` maal `d` spaties, en een regelovergang.
   Wellicht ten overvloede, een regel met een stel spaties en TAB's, en dan verder alleen `//`-commentaar, komt als `tab` maal `d` spaties in de uitvoerfile.
   We nemen aan dat de accolades netjes gepaard zijn.
   En op welke positie staat een accolade, als dit het eerste symbool is dat op een regel wordt afgedrukt? We spreken af dat als dit een openings-accolade is hiervoor nog de "oude" diepte wordt gebruikt, en voor een sluit-accolade de "nieuwe". We krijgen dus (waarbij een punt (.) een spatie voorstelt; `tab` = 3, `d` = 2):
   ```
   ......if ( x == y )
   ......{
   .........z = 0;
   ......}
   ```
3. Elk optreden van een drietal letters moet worden geteld.
   Vraag aan het begin drie verschillende kleine letters aan de gebruiker (net zolang tot dat gelukt is). Tel hoe vaak deze drie letters, in die volgorde, in de file voorkomen.
   Hierbij matcht een hoofdletter in de tekst de bijbehorende kleine letter. Optredens binnen strings en binnen commentaar tellen ook mee. Er mogen hierbij geen strings worden gebruikt.
4. Het Collatz-vermoeden moet worden gecontroleerd.
   Voor elk geheel getal > 0 uit de invoerfile wordt gekeken of het Collatz-vermoeden waar is voor dat getal; zie ook sheet 17 van het derde college. Op het scherm wordt afgedrukt wat het aantal iteraties is

bij 1 te komen, of het nummer van de iteratie waarvan het resultaat boven `INT_MAX` (gebruik `include <climits>`) uitkomt. Als dit laatste gebeurt, wordt dit erbij vermeld.
Elke directe opeenvolging van cijfers in de invoerfile wordt als een geheel getal opgevat. Neem aan dat ze alle kleiner dan of gelijk aan `INT_MAX` zijn. Zo bevat `123abcd-"qqq 5"+++uvw-77.88ddd//vb5656` de gehele getallen 123, 5, 77 en 88. Het maakt verder ook niet uit of een getal al dan niet binnen een string staat, het telt gewoon mee. Getallen binnen commentaar worden niet gedaan.

Na afloop moet worden meegedeeld of de accolades in de invoerfile goed "gepaard" waren. Het kan om twee redenen mis zijn gegaan: te veel of te weinig sluitaccolades.
Ook wordt dan het aantal optredens van de drie letters genoemd.

Ter verdere inspiratie, zie het vijfde werkcollege, ook voor voorbeeldfiles. Let op: files van websites kopiëren door met rechter muisknop op de links te klikken, anders (met markeer-copy-paste) gaan spaties/tabs wellicht fout!

Opmerkingen

- We nemen aan dat de gebruiker zo vriendelijk is verder geen fouten te maken bij het invoeren van gegevens. Als een getal gevraagd wordt, geeft hij/zij een getal.
- Gebruik de regelstructuur: elke regelovergang in een bestand bestaat uit een LineFeed (`\n`) (in UNIX) of een CarriageReturn gevolgd door een LineFeed (`\r\n`) (in Windows). Normaal gesproken gaat dit "vanzelf" goed. We nemen aan dat er voor het EndOfFile-symbool (wat dat ook moge zijn) een regelovergang staat.
- Alleen voor de namen van de files mag een array (of string) gebruikt worden; voor het lezen en verwerken van de tekst is slechts het huidige karakter en enige kennis over de voorgaande karakters nodig — zie boven. Alleen de headerfiles `iostream` en `fstream` mogen gebruikt worden (en `string` voor de filenamen; denk in dat geval aan het gebruik van `c_str`; en `climits` voor `INT_MAX`). Uit een file mag alleen met `invoer.get (...)` gelezen worden, vergelijk Hoofdstuk 3.7 uit het dictaat, gedeelte "aantekeningen bij de hoorcolleges". Binnen de hoofdloop van het programma staat bij voorkeur maar één keer een `get`-opdracht, vergelijk het voorbeeldprogramma uit dit hoofdstuk (daar staat twee keer `get`, één maal vóór de loop, uiteraard). Karakters mogen niet worden teruggezet in de oorspronkelijke file.
  Schrijf zelf functies die testen of een karakter een cijfer is, etcetera. Er mogen geen andere functies dan die uit `fstream` gebruikt worden, en `c_str`.
- Denk aan het infoblokje dat aan begin op het scherm verschijnt. Gebruik enkele geschikte functies, bijvoorbeeld voor infoblokje, inlezen gegevens van de gebruiker, omkeren van het getal, en coderen en decoderen van een file (zie de tips bij het vijfde werkcollege). Globale variabelen zijn streng verboden. Ruwe indicatie voor de lengte van het C++-programma: circa 250 regels.

Uiterste inleverdatum: maandag 18 oktober 2021, 17:00 uur.

Manier van inleveren:

1. Digitaal de C++-code inleveren: stuur een email naar `pm@liacs.leidenuniv.nl`.
   Stuur geen executable's, lever alleen de C++-file digitaal in! Noem deze bij voorkeur zoiets als `kaagrutte2.cc`, dit voor de tweede opdracht van het duo Rutte-Kaag. De laatst voor de deadline ingeleverde versie wordt nagekeken.
2. En *ook* een papieren versie van het verslag (inclusief de C++-code) deponeren in de speciaal daarvoor bestemde doos "Programmeermethoden" bij kamer 159 van het Snellius-gebouw.
   Overal duidelijk datum en namen van de (maximaal twee) makers vermelden, in het bijzonder als commentaar in de eerste regels van de C++-code. Lees bij het zesde werkcollege hoe het verslag eruit moet zien en wat er in moet staan.

Te gebruiken compiler: als hij maar C++ vertaalt; het programma moet in principe zowel op een Linux-machine (met `g++`) als onder Windows met Code::Blocks draaien. Test dus in principe op beide systemen! Normering: verslag 1; layout 1; commentaar 1; overzichtelijkheid/modulariteit 2; werking 5. Eventuele aanvullingen en verbeteringen: lees deze WWW-bladzijde: `www.liacs.leidenuniv.nl/~kosterswa/pm/op2pm.php`.

www.liacs.leidenuniv.nl/~kosterswa/pm/op2pm.php

- werk aan de tweede programmeeropgave — de deadline is op maandag 18 oktober 2021, 17:00 uur
- lees Savitch Hoofdstuk 3 en 4, en 12.1/2
- lees dictaat Hoofdstuk 3.6, 3.7 en 4.1
- maak opgaven 11/17 uit het opgavendictaat
- doe het `vierde werkcollege`
- `www.liacs.leidenuniv.nl/~kosterswa/pm/`

Programmeermethoden

Functies — vervolg

Walter Kosters en Jonathan Vis

week 5: 4–8 oktober 2021

www.liacs.leidenuniv.nl/~kosterswa/pm/

Een eenvoudige `void`-functie:

```
// kort infoblokje
void infoblokje ( ) {
  cout << "Hallo allemaal ..." << endl;
  cout << "Enzovoorts ..."<< endl;
}//infoblokje

int main ( ) {
  ...
  infoblokje ( );
  ...
}//main
```

Verwissel = swap

Een functie die zijn parameters omwisselt:

```
void wissel (int & x, int & y) { call by reference: &
  int hulp = x;
  x = y;
  y = hulp;
}//wissel
```

Met aanroep:

```
int a = 33, x = 88;
cout << a << " en " << x << endl; // 33 en 88
wissel (a,x);
cout << a << " en " << x << endl; // 88 en 33
```

Tel `x` en `y` op (als return-waarde resp. in `z`):

```
int telop (int x, int y) {
  return x + y;
}//telop  aanroep: som = telop (8,13);
void telop2 (int x, int y, int & z) {
  z = x + y;
}//telop2 aanroep: telop2 (8,13,som);
```

Tel breuken $\frac{x1}{x2}$ en $\frac{y1}{y2}$ op in $\frac{z1}{z2}$:

```
void telbreukenop (int x1, int x2, int y1, int y2,
                   int & z1, int & z2) {
  z1 = x1 * y2 + y1 * x2;
  z2 = x2 * y2;
}//telbreukenop
```

vierde werkcollege

Gevraagd: zet de cijfers van een getal op aparte regels, het laatste cijfer eerst:

```
// hier moet commentaar staan, dus:
// zet cijfers van getal > 0 omgekeerd op aparte regels

void cijfers (int getal) {
  while ( getal != 0 ) {
    cout << getal % 10 << endl; // of tel de cijfers ...
    getal = getal / 10;
  }//while
}//cijfers
```

Neem aan dat `getal` minstens 0 is.

Een getal omkeren (dus 196 → 691) gaat analoog!

Geen strings!

Top-down en bottom-up

Bij top-down maak je een functie als je hem nodig hebt, bij bottom-up bedenk je hem voordat je hem nodig hebt.

Voorbeeld: machtverheffen, $y = x^7$. Bij bottom-up gebruik je `pow` uit `<cmath>` of uit `"zelf.h"`, bij top-down maak je:

```
// bereken x tot de n-de voor n >= 0
int machtsverheffen (int x, int n) {
  int i;  // tellertje
  int res = 1;  // om resultaat in op te bouwen
  for ( i = 1; i <= n; i++ ) res = res * x;
  return res;
}//machtsverheffen
```

Daarna zet je dit alsnog in `"zelf.h"`.

PS En met 5 vermenigvuldigingen x^{15} berekenen?

```
void john (int x, int y) { ... }//john

int paul (double x, bool b) { ... }//paul

void george ( ) { ... }//george

bool ringo (int & getal) { ... }//ringo

int main ( ) { ... }//main
```

De functie `george` mag de functies `george` ("recursie"), `john` en `paul` gebruiken = aanroepen, maar *niet* de functie `ringo`! Wil je dat toch, dan moet je boven `george` een prototype `bool ringo (int & getal);` toevoegen.

Dus bij

```
bool ringo (int & getal); // prototype ringo

void george ( ) { ... }//george

bool ringo (int & getal) { ... }//ringo

int main ( ) { ... }//main
```

mogen `ringo` en `george` elkaar beide(n) aanroepen! Dankzij het prototype van `ringo` (let op de ;) mag `george` de eigenlijk verderop gedefinieerde `ringo` toch aanroepen.

En om misverstanden te vermijden: functies worden binnen functies aangeroepen, maar *na elkaar* en niet binnen elkaar gemaakt.

Functies hebben verschillende soorten parameters:

- globaal — gelden overal
- locaal — gelden alleen binnen een functie (of . . .)
- formeel — staan in functie-heading
- actueel — bij aanroep van een functie
- call by value — geef waarde door
- call by reference — geef (adres van) variabele door

Voorbeeld — call by value

```
void hoogop (int x) { x = x + 10; cout << x; }//hoogop

void maaknul (int t) { t = 0; cout << t; }//maaknul
```

```
int x, m, q;
x = 7; hoogop (x); cout << x;          17 7
m = 3; hoogop (m+8); cout << m;        21 3
q = 5; maaknul (q); cout << q;         0 5
maaknul (42);                          0
```

Er wordt alleen een *waarde* doorgegeven, en wel van de *actuele* parameter aan de *formele* parameter; er wordt dus een "lokale kopie" gemaakt, wat tijd en ruimte kost.

Voorbeeld — call by reference

```
void hoogop (int & x) { x = x + 10; cout << x; }//hoogop

void maaknul (int & y) { y = 0; cout << y; }//maaknul

  int x, m, q;
  x = 7; hoogop (x); cout << x;          17 17
  m = 3; hoogop (m+8); // VERBODEN!!!
  q = 5; maaknul (q); cout << q;         0 0
  maaknul (42);  // VERBODEN!!!
```

Er wordt nu een *adres* (een *pointer*) doorgegeven. De *actuele* parameter kan nu wel veranderen. De *actuele* parameter mag geen "rare" expressie als `m+8` of `42` zijn. Er wordt alleen een *adres* gekopieerd.

Globale variabelen

En met een globale variabele erbij:

```
int globaal; // globale variabele, geldt overal, vermijden

int doewat (char kar, double & getal) {
  // kar is call by value, getal call by reference
  int locaal; // locale variabele, geldt binnen doewat
  ...
}//doewat

void nogeen ( ) {
  double locaal; // locale variabele, geldt binnen nogeen
  cout << doewat (globaal,locaal) << endl;
  // waarde van globaal gaat naar kar (met casting)
  // locaal "is" hetzelfde als getal uit doewat
}//nogeen
```

```
int a; int b;
void kwadraat (int   a) { // call by value
  a = pow (a,2); // uit <cmath>, oftewel a * a
  b++;
  cout << "0: " << a << " en " << b << endl;
}//kwadraat
```

Nu doen we:

(1)
```
a = 5; b = 13; kwadraat (a);
    cout << "1: " << a << " en " << b << endl;
```
(2)
```
a = 2; b = 7;  kwadraat (b);
    cout << "2: " << a << " en " << b << endl;
```

Dat levert

```
0: 25 en 14
1: 5 en 14
```

```
0: 49 en 8
2: 2 en 8
```

```
int a; int b;
void kwadraat (int & a) { // call by reference
  a = pow (a,2); // uit <cmath>, oftewel a * a
  b++;
  cout << "0: " << a << " en " << b << endl;
}//kwadraat
```

Nu doen we:

```
(3) a = 5; b = 13; kwadraat (a);
    cout << "3: " << a << " en " << b << endl;
(4) a = 2; b = 7;  kwadraat (b);
    cout << "4: " << a << " en " << b << endl;
```

Dat levert

```
  0: 25 en 14          0: 50 en 50
  3: 25 en 14          4: 2 en 50
```

```
void alias (int r, int & s) {
  int t;
  t = 3;
  r = r + 2;
  s = s + r + t;
  t = t + 1;
  r = r - 3;
  cout << r << " " << s << " " << t << endl;
}//alias
...
  t = 12; alias (t,t); cout << t << endl;
```

t = s	r	t'
12		
	12	?
		3
	14	
29		
		4
	11	

Dit levert: `11 29 4` en 29.

En met een `&` voor `r`: 28 28 4 en 28.

Een functie mag zichzelf (in)direct aanroepen: recursie.

```
int som (int n) { // berekent 1 + 2 + ... + n   versie 1
  int i, res = 0;
  for ( i = 1; i <= n; i++ ) res += i;
  return res;
}//som

int somrecursief (int n) { // idem, recursief   versie 2
  if ( n == 0 ) return 0;
  else return n + somrecursief (n-1);
}//somrecursief

int somslimGauss (int n) { // en nog eens ...   versie 3
  return ( n * ( n + 1 ) ) / 2;
}//somslim
```

De ggd kan ook recursief berekend worden:

```
int ggdrecursief (int x, int y) {
  if ( y == 0 ) return x;
  else return ggdrecursief (y,x % y);
}//ggdrecursief
```

Je gebruikt eigenlijk:

$$\text{ggd}(x,y) = \begin{cases} x & \text{als } y = 0 \\ \text{ggd}(y, x \bmod y) & \text{als } y \neq 0 \end{cases}$$

Voor meer over recursie, zie later.

Tweede prog'opgave: Netjes

Voor de tweede programmeeropgave moet je een C++-programma schrijven dat een gegeven file netjes ingesprongen afdrukt, en daarbij //-commentaar verwijdert:

```
␣␣if␣(␣leeftijd␣␣>␣6171/97␣)␣␣{{␣//␣commentaar!
cout␣<<␣"Oud";//␣64␣of␣niet?

if␣(␣leeftijd␣␣>␣6171/97␣)␣␣{{␣
␣␣␣␣␣␣cout␣<<␣"Oud";
```

Hier is ␣ een spatie. En "tab = 3".

Hoe vaak komt `mag` voor? Klopt Collatz voor 6171?

www.liacs.leidenuniv.nl/~kosterswa/pm/op2pm.php

Tweede programmeeropgave

1. commentaar wegwerken,
   test dat goed met voorbeeldfiles;
   gebruik zo weinig mogelijk `put`'s en `get`'s

2. inspringen

3. drietal letters zoeken

4. daarna de Collatz-controle

5. en tot slot details . . . en het verslag

Houd het kort! Gebruik geschikte functies; zie de tips:

www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc4.php
www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc5.php
www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc6.php

Met `char kar = invoer.get ( );` probeer je het eerste karakter uit `invoer` te halen; met `invoer.eof ( )` verifieer je of je aan het einde van de `invoer` stond.

Met `uitvoer.put (kar);` schrijf je `kar` achteraan de `uitvoer`. Dat kan ook met `uitvoer « kar;` .

Let op het verschil tussen `kar = invoer.get ( );` en `invoer » kar;` . Die tweede slaat "whitespace" (waaronder spaties en regelovergangen) over!

En `kar = cin.get ( );` wacht op het eerste karakter vanaf het toetsenbord (met ooit een "enter").

Getal opbouwen

Stel dat iemand karakters (`char`'s, waaronder cijfers) op je afstuurt, en je daar een getal van moet maken. Hoe doe je dat?

Gebruik `int getal = 0;`, en herhaal:

```
if ( '0' <= kar && kar <= '9' )
  getal = 10 * getal + ( kar - '0' );
else
  ...
```

```
qwerty7392abc   de---12fghijklmnopq
        ↑
```

`getal` is 73 en `kar` is '9'

`getal` wordt 739

Deze `void`-functie manipuleert een file `invoer`:

```
void manipuleer (ifstream & invoer) {
  char prevkar = '\n', kar = invoer.get ( );
  while ( ! invoer.eof ( ) ) {
    if ( ... )                  // GEEN get's,
      ...                       // GEEN while's
    if ( ... )                  // GEEN strings,
      ... put ...               // ...
    prevkar = kar;
    kar = invoer.get ( );
  }//while
}//manipuleer
```

Deze `void`-functie telt niet-lege regels in een file `invoer`:

```
void telNietLegeRegels (ifstream & invoer, int & tel) {
  char prevkar = '\n', kar = invoer.get ( );
  while ( ! invoer.eof ( ) ) {
    if ( prevkar != '\n' && kar == '\n' )
      tel++;
    prevkar = kar;
    kar = invoer.get ( );
  }//while
}//telNietLegeRegels
```

Het aantal wordt opgeteld bij de oorspronkelijke "oude" waarde van `tel`. De actuele parameter bij die aanroep wordt dus gewijzigd.

Het is meestal verstandig invoer, rekenwerk en uitvoer door verschillende functies te laten verrichten.

Deze `int`-functie telt ook niet-lege regels in een file `invoer`:

```
int telNietLegeRegels (ifstream & invoer) {
  char prevkar = '\n', kar = invoer.get ( );
  int tel = 0; // locaal tellertje
  while ( ! invoer.eof ( ) ) {
    if ( prevkar != '\n' && kar == '\n' )
      tel++;
    prevkar = kar;
    kar = invoer.get ( );
  }//while
  return tel;                    // <======= int functie!
}//telNietLegeRegels
```

Denk aan het openen en sluiten van de file(s).
Soms problemen met regelovergangen Windows/Linux:
`\r\n` respectievelijk `\n`. `(... ios::in | ios::binary ...)`

- werk aan de tweede programmeeropgave — de deadline is op maandag 18 oktober 2021, 17:00 uur
  vorm tweetallen!

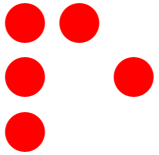
- lees Savitch Hoofdstuk 3 en 4

- lees dictaat Hoofdstuk 3.6 en 3.7

- kijk de video's

- www.liacs.leidenuniv.nl/~kosterswa/pm/

Programmeermethoden



Functies & Life

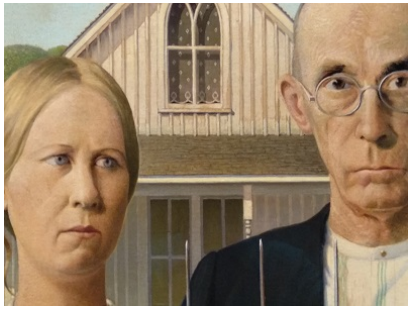
Walter Kosters en Jonathan Vis

week 6: 11–15 oktober 2021

www.liacs.leidenuniv.nl/~kosterswa/pm/

Voor de tweede programmeeropgave moet je een C++-programma schrijven dat een gegeven file netjes ingesprongen afdrukt, en daarbij //-commentaar verwijdert:

```
␣␣if␣(␣leeftijd␣␣>␣6171/97␣)␣␣{{␣//␣commentaar!
␣cout␣<<␣"Oud";//␣64␣of␣niet?

if␣(␣leeftijd␣␣>␣6171/97␣)␣␣{{␣
␣␣␣␣␣␣cout␣<<␣"Oud";
```



Hier is ␣ een spatie. En "tab = 3".

Hoe vaak komt `mag` voor? Klopt Collatz voor 6171?

www.liacs.leidenuniv.nl/~kosterswa/pm/op2pm.php

1. commentaar wegwerken,
   test dat goed met voorbeeldfiles;
   gebruik zo weinig mogelijk `put`'s en `get`'s
2. inspringen, idem; denk aan "pairing"
3. drietal letters zoeken
4. daarna de Collatz-controle (`INT_MAX`!)
5. en tot slot details . . . en het verslag

↙ $\leqslant \approx$ 250 regels

Houd het kort! Gebruik geschikte functies; zie de tips:

www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc4.php
www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc5.php
www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc6.php

Stel dat iemand karakters (`char`'s, waaronder cijfers) op je afstuurt, en je daar een getal van moet maken. Hoe doe je dat?

Gebruik `int getal = 0;`, en herhaal:

```
if ( '0' <= kar && kar <= '9' )
  getal = 10 * getal + ( kar - '0' );
else
  ...
```

`qwerty7392abc  de---12fghijklmnopq`

↑

`getal` is 73 en `kar` is '9'

`getal` wordt 739

Deze `int`-functie telt het aantal keer dat een 'd' direct door een 'e' gevolgd wordt in een al geopende file `invoer`:

```
int telDE (ifstream & invoer) {
  char prevkar = '\n', kar = invoer.get ( );
  int tel = 0; // locaal tellertje
  while ( ! invoer.eof ( ) ) {
    if ( prevkar == 'd' && kar == 'e' )
      tel++;
    prevkar = kar;
    kar = invoer.get ( );
  }//while
  return tel;                  // <======= int functie!
}//telDE
```

- weer in LaTeX, gebruik mooi.tex

- voorbeeldfile om werking te illustreren (`tab` = 3):
  `abbbbc{{6171//def` ⟶ `abbbbc{{6171`
  `zes}}` ␣␣␣␣␣␣`zes}}`

- iets over het Collatz-vermoeden, en enkele vragen beantwoorden (-7, ...)

- urentabel

week	1	2	3	totaal
Mark	6	4	6	16
Sigrid	6	3	5	14
totaal	12	7	11	30

- zie www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc6.php

Bereken $A(n) = 1/2 + 2/4 + 3/8 + \ldots + n/2^n$:

```
double sommetje (int n) {
  int teller;       // teller van teller-de term
  int noemer = 1;   // en de noemer daarvan
  double som = 0;   // de (deel)som
  for ( teller = 1; teller <= n; teller++ ) {
    noemer *= 2;   // noemer = noemer * 2;
    som += static_cast<double>(teller) / noemer;
  }//for
  return som;
}//sommetje
```

(Eigenlijk kun je beter met $n/2^n$ beginnen . . .)

www.liacs.leidenuniv.nl/~kosterswa/pm/handouts.php

```
void test (int x, int & y) {
  int z = 9; x = 5; y = 6; z = 7;
}//test
```

Steeds eerst `x = 1; y = 2; z = 3;` , en daarna `x`, `y` en `z` afdrukken:

a. `test (x,y);` geeft 1, 6, 3

b. `test (y,x);` geeft 6, 2, 3

c. `test (1,z);` geeft 1, 2, 6

d. `test (z,1);` mag niet, er moet een "variabele" (l-value) op de tweede plek staan!

e. `test (z,x);` geeft 6, 2 ,3

Eigenlijk maakt de functie alleen zijn tweede variabele 6.

```
int f (int x, int y) { x--; return x * y; }//f
int g (int a, int b) {
  int x = 3; b += x; a--; a = f (a,b) + f (a,a);
  cout << x << a << b << endl; return a + x - 2; }//g
```

a. `x = 6; y = 16; cout << g(x,y); cout << x << y << endl;`
levert: 3, 96, 19 97, 6, 16

b. `int G (int a, int b) { return (a-2)*(a+b+2) + 1; }//G`

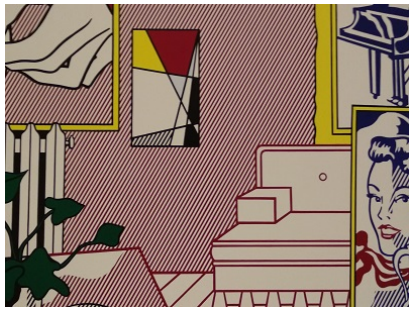
c. Als **a**, met vier &'s
Als eerst `f (a,b)` wordt geëvalueerd: 76, en `a` (dus `x`) is nu 4. Dan `f (a,a)`, geeft 9, en `a` (dus `x`) is nu 3. Dat levert: 3, 85, 19, 86, 85, 19. Met eerst `f (a,a)`: 3, 73, 19, 74, 73, 19. De volgorde is onduidelijk in C++.

```
int peter (int r, int s) { s--; return r+s+2; }//peter
int ellen (int p, int q) {
  int a = 7; p++; q -= 2;
  for ( a = 2; a < q; a++ ) p = p + peter (p,q);
  cout << a << p << q << endl; return a+p+q; }//ellen
```

a. `a = 2; b = 6; cout << ellen (a,b); cout << a << b << endl;`

b. Idem, met vier &'s.

c. Als **b**, nu met `p = p + peter (q,p);` .



```
int peter (int r, int s) { s--; return r+s+2; }//peter
int ellen (int p, int q) {
  int a = 7; p++; q -= 2;
  for ( a = 2; a < q; a++ ) p = p + peter (p,q);
  cout << a << p << q << endl; return a+p+q; }//ellen
```

a. `a = 2; b = 6; cout << ellen (a,b); cout << a << b << endl;`

a	b	p_{ellen}	q_{ellen}	a_{ellen}
2	6			
		2	6	7
		3		
			4	
				2
		11 (*)		
				3
		27 (*)		
				4

(*) `peter (3,4)` geeft 8, en `peter (11,4)` geeft 16.

Afgedrukt wordt: 4, 27, 4 35, 2, 6

```
int peter (int & r, int & s) { s--; return r+s+2; }//peter
int ellen (int & p, int & q) {
  int a = 7; p++; q -= 2;
  for ( a = 2; a < q; a++ ) p = p + peter (p,q);
  cout << a << p << q << endl; return a+p+q; }//ellen
```

b. `a = 2; b = 6; cout << ellen (a,b); cout << a << b << endl;`

$a = p_{ellen}$	$b = q_{ellen}$	a_{ellen}
2	6	
		7
3		
	4	
		2
	3 (*)	
11 (*)		
		3

(*) `peter (p,q)` geeft 8, en laagt `q` met 1 af. Loop stopt!

Afgedrukt wordt: 3, 11, 3 17, 11, 3

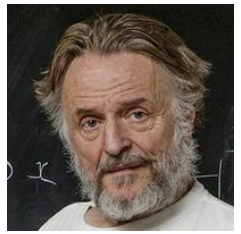
```
int peter (int & r, int & s) { s--; return r+s+2; }//peter
int ellen (int & p, int & q) {
  int a = 7; p++; q -= 2;
  for ( a = 2; a < q; a++ ) p = p + peter (q,p); // <-- volgorde anders
  cout << a << p << q << endl; return a+p+q; }//ellen
```

c. `a = 2; b = 6; cout << ellen (a,b); cout << a << b << endl;`

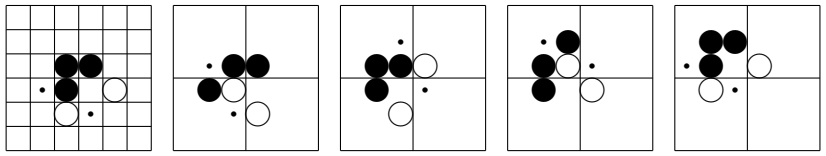
$a = p_{ellen}$	$b = q_{ellen}$	a_{ellen}
2	6	
		7
3		
	4	
		2
10 (*)		
		3
24 (*)		
		4

(*) Stel eerst: `peter (q,p)` geeft 8, en laagt `p` met 1 af naar 2. Analoog, de tweede keer: `p` wordt 24.
Afgedrukt wordt: 4, 24, 4 32, 24, 4
Maar (bij andere volgorde) kan a ook 25/26/27 zijn . . .

Life is een "cellulaire automaat", in 1970 bedacht door John Horton Conway (1937–2020).



In een 2-dimensionaal oneindig groot rooster beginnen we met een eindig aantal levende vakjes oftewel cellen. Een levend vakje met minder dan 2 of meer dan 3 buren (van de 8) gaat dood, met precies 2 of 3 levende buren overleeft het. In een dood vakje met precies 3 levende buren ontstaat leven. Dit leidt tot de volgende generatie. Let erop dat dit voor alle vakjes tegelijk gebeurt.

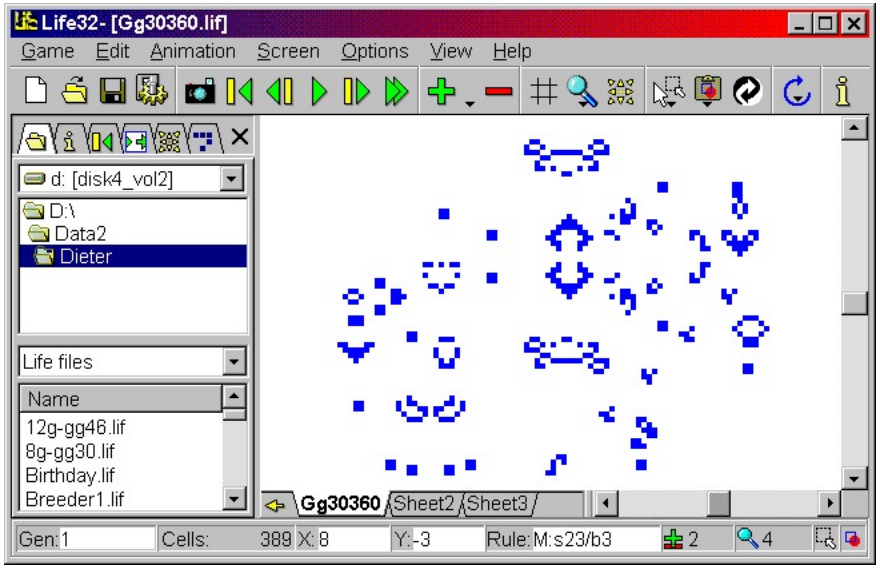


● levend
○ gaat dood
· (komt tot leven)

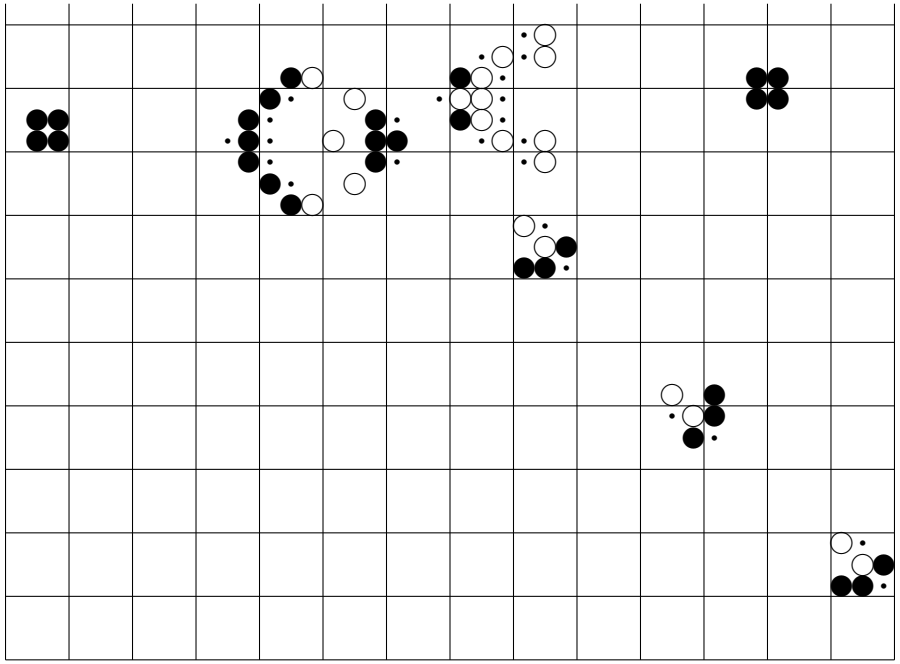
Dit patroon heet glider.

- Wiki: http://www.conwaylife.com/wiki/

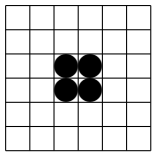
- Programma (Windows):
  https://github.com/JBontes/Life32 (Binary)

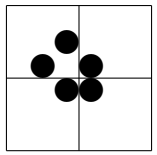


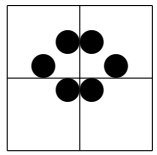
In 1970 wonnen onderzoekers van het M.I.T. in Boston $50 met een beginconfiguratie waarbij het aantal levende cellen groter en groter wordt: Gosper's glider gun, die elke dertigste generatie een nieuwe glider afvuurt:



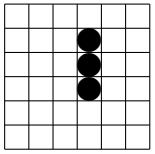
Een stilleven is een Life-configuratie die niet verandert:

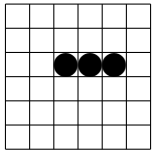

blok

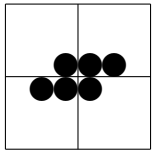

boot

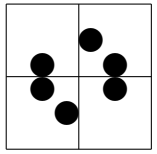

bijenkorf

En een oscillator repeteert met een zekere periode (stilleven is een periode-0 oscillator):

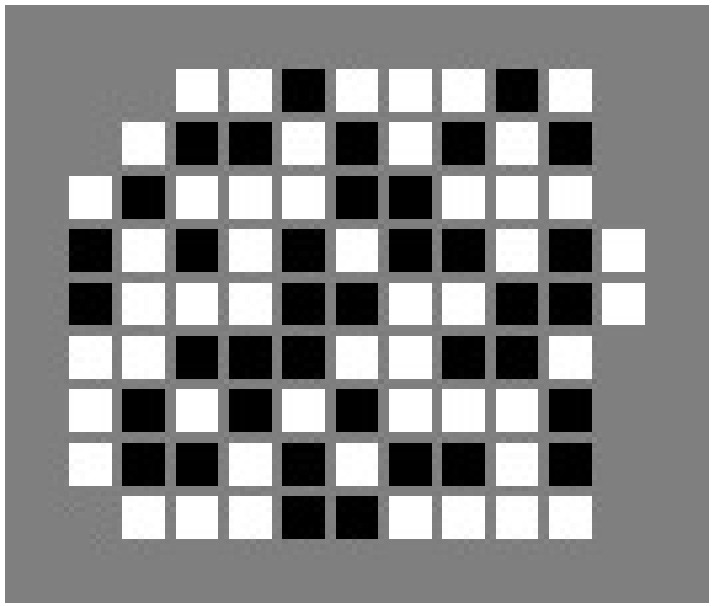

1 blinker


2


1 pad


2

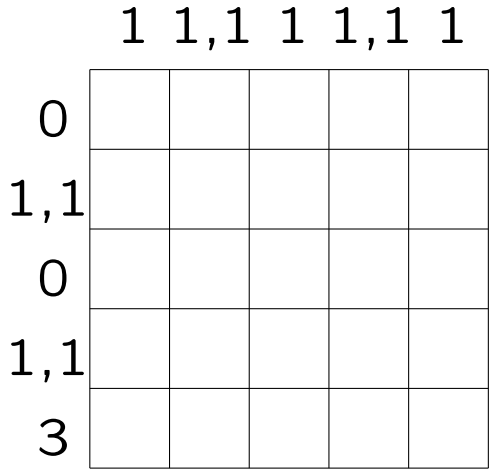
Een wees = orphan is een life-(deel)patroon dat nooit kan ontstaan tijdens de ontwikkeling vanuit een beginpatroon. Minder algemeen, een Hof van Eden heeft geen "ouder".

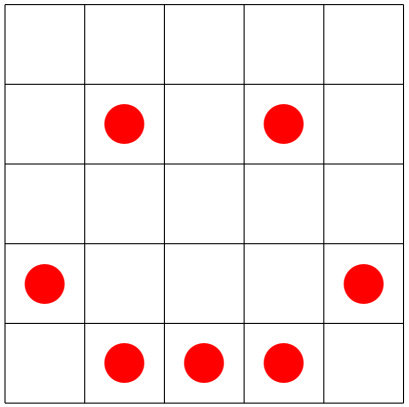


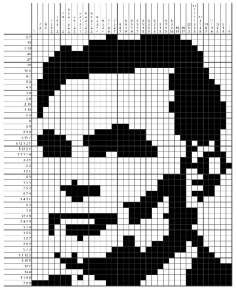
Steven Eker, 2017

Een breeder is een life-configuratie die glider guns produceert:

Japanse puzzels (Nonogrammen) zien er zo uit:







Naast iedere rij en boven iedere kolom staan in volgorde de lengtes van aaneengesloten series rode blokjes.

Voor Life/Nonogram: 2-dimensionale arrays (matrices)!

- werk aan de tweede programmeeropgave — de deadline is op maandag 18 oktober 2021, 17:00 uur

- maak opgaven tot en met 25 uit het opgavendictaat

- www.liacs.leidenuniv.nl/~kosterswa/pm/

Programmeermethoden

Object-geOriënteerd Programmeren & arrays

Walter Kosters en Jonathan Vis

week 7: 18–22 oktober 2021

www.liacs.leidenuniv.nl/~kosterswa/pm/

```
string filenaam;  // gebruik <string>
ifstream invoer;  // gebruik <fstream>
...
cin >> filenaam;
invoer.open (filenaam.c_str ( )); // (*)
if ( invoer.fail ( ) ) {
  cout << filenaam << " niet te openen" << endl;
  return 1;  // of exit (1);
}//if
```

In bovenstaand programma maken we een object `filenaam` van klasse `string` (voor de naam van de file) en een object `invoer` van klasse `ifstream` (voor de file). In regel `(*)` koppelen we ze, door de methode `open` te gebruiken. Sinds C++11 mag hier ook `invoer.open (filenaam);` staan.

C++ is — in tegenstelling tot C — een object-georiënteerde (OO) programmeertaal, net als Java.

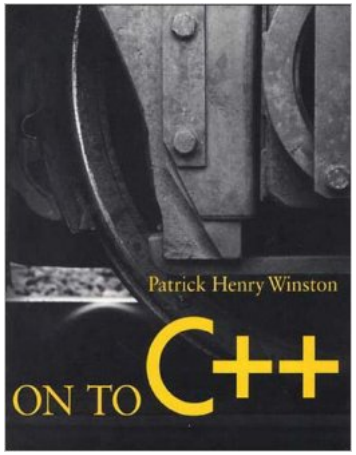
In een OO-programma hebt je objecten (zoals Adam, Eva; Bonzo) van verschillende klassen (Mens; Hond). Klassen hebben hun eigen methoden (praten, slapen, blaffen).

Een voorbeeld: de klasse `double`. Met `double x, y;` maak je twee objecten (variabelen) van deze klasse. En `cout << x;` vraagt `x` zich af te laten drukken. En `x = x - y;` vraagt `x` zich met de waarde van `y` af te lagen.

Denk ook aan files met methodes (member-functies) als `get`, `put`, `eof`, `open` en `close`.

Een eerste klasse — 1

```
class wagon {
  public:
    double hoogte, breedte, lengte;
    double inhoud ( ) {  // member-functie
      return hoogte * breedte * lengte;
    }//inhoud
};//wagon; let op de ;
...
wagon bert;
bert.hoogte = 3.5;
bert.breedte = 4.0;
bert.lengte = 20.5;
cout << "Inhoud: " << bert.inhoud ( ) << endl;
```



Hier is `bert` een object van klasse (= type) `wagon`.

Een eerste klasse — 2

Een object `ernie` van klasse `wagon` bestaat uit drie double's, die zijn hoogte, breedte en lengte aanduiden.

En je kunt (via de methode `inhoud`) om zijn inhoud vragen. Deze functies worden eenmalig opgeslagen, niet in ieder object opnieuw.

Let op de punt-notatie: `ernie.breedte`. En voor functies (methoden) `ernie.inhoud ( )`.

Tevens kan bestaan (overloading van `inhoud` en `lengte`):

```
class tanker {
  public:
    double straal, lengte;  // zelfde namen
    double inhoud ( ) {     // als zo-even!
      return straal * straal * lengte;
    }//inhoud
};//tanker
```

```
class wagon {
  public:
    double hoogte, breedte, lengte;
    double belasting (double);
      // functie-prototype van deze methode
};//wagon
double wagon::belasting (double percentage) {
  return percentage * breedte * lengte;
}//wagon::belasting
```

Hierbij is `::` de binary scope resolution operator.
Met `ernie` een object van klasse `wagon` (dus `wagon ernie;`):

```
cout << "Belasting: "
     << ernie.belasting (0.5) << endl;
```

Het benutten van de (member-)variabelen van een object gaat meestal met speciaal geschreven functies: reader (*getter*, *accessor*) en writer (*setter*, *mutator*) methodes.

```
class tanker { ... als vroeger ...
  double geefstraal ( ) {  // reader
    return straal;
  }//geefstraal
};//tanker
```

Gebruik nu `zeppo.geefstraal ( )` in plaats van `zeppo.straal` (met `zeppo` van type `tanker`). Analoog writer's.
Een uitbreiding voor `tanker`:

```
double tanker::geefdiameter ( ) {  // reader
  return 2.0 * straal;
}//geefdiameter
```

Met behulp van reader's en writer's kun je (member-)variabelen van een object afschermen/verbergen:

```
class tanker {
  public:
    double geefstraal ( ) {  // reader
      return straal;
    }//geefstraal
    void maaklang (double t) {  // writer
      lengte = t;
    }//maaklang
  private:
    double straal, lengte;
};//tanker
```

Nu mag `chico.straal` zelfs niet meer gebruikt worden; het *moet* via `chico.geefstraal ( )` (met `chico` van type `tanker`). En je *moet* nu `chico.maaklang (42.1);` doen in plaats van `chico.lengte = 42.1;`.

```
class tanker {
  public:
    double straal, lengte;
    tanker ( ) {
      straal = 1.0; lengte = 37.0;
    }//default constructor
    tanker (double s, double t) {
      straal = s; lengte = t;
    }//constructor
};//tanker
```

Als je nu een nieuwe variabele maakt van klasse `tanker` kun je die meteen initialiseren:

```
tanker harpo;                  // met default constructor
tanker groucho (7.0,12.12); // met andere constructor
```

Een constructor wordt "nooit" direct aangeroepen, maar automatisch gebruikt bij het ontstaan van objecten.

De klasse `personenwagon` wordt afgeleid (= derived) van de ouder (= superklasse) `wagon`:

```
class personenwagon : public wagon {
  // we erven "alles" van wagon
  public:
    // default constructor:
    personenwagon ( ) { passagiers = 0; }
    personenwagon (int aantal) {
      passagiers = aantal;
    }//constructor
    int hoeveel ( ) { // reader
      return passagiers;
    }//hoeveel
  private:
    int passagiers;
};//personenwagon
```

Ook multiple inheritance/overerving is mogelijk:

```
class gehakt : public dier, eten { ... };
```

Stel we hebben een klasse `voertuig`, met variabelen `gewicht` en `maxsnelheid`, en een methode `belasting ( )`. Er zijn afgeleide klassen `fiets` (met eigen methode `belasting ( )`) en `auto` (met een extra variabele `soort`).

Met `rijwiel` van type `fiets` mag je gebruik maken van `rijwiel.belasting ( )`. Je krijgt dan de belasting speciaal voor een fiets. Als je toch de belasting als voor een voertuig wilt laten berekenen: `rijwiel.voertuig::belasting ( )`.

Als je de constructor voor `fiets` "aanroept", wordt automatisch eerst die voor `voertuig` uitgevoerd.

Stel we willen met gehele getallen van "willekeurige" lengte werken, zoals 1234567891011121314151617181920. Grote getallen dus. We maken daartoe een klasse `gg` met methoden als `drukaf ( )`, `maak (int m)`, `kopie (gg & getal)` en `telop (gg & getal)`.

Je kunt dan een programma schrijven als

```
gg x; gg y;                                  // int x; int y;
x.maak (1); y.kopie (x);                     // x = 1; y = x;
for ( int i = 1; i <= 1000; i++ ) {
  x.telop (y);                               // x = x + y;
  y.kopie (x);                               // y = x;
  x.drukaf ( );                              // cout << x;
}//for
```

Dit berekent uiteindelijk 2^{1000} (het kan anders en beter).

- polymorfisme en late binding
- kopiëren van objecten (“diepe kopie”)
- destructoren
- private, protected, public
- operatoren bijdefiniëren
- this-pointer: `wagon* p = this;`

Een array is een geordend rijtje variabelen van hetzelfde type, bijvoorbeeld een vector met 10 "reële" getallen: na

```
double A[10];
```

heb je 10 `double`'s, namelijk

`A[0]`, `A[1]`, `A[2]`, `A[3]`, `A[4]`,
`A[5]`, `A[6]`, `A[7]`, `A[8]` en `A[9]`.

Er zijn ook 2-dimensionale arrays: *matrices* (Life! Nonogram!).

Naamgeving: `A[4]` is een array-element (het vierde, of eigenlijk het vijfde), `4` de bijbehorende array-index.

Maak eerst een constante:

```
const int MAX = 100;
```

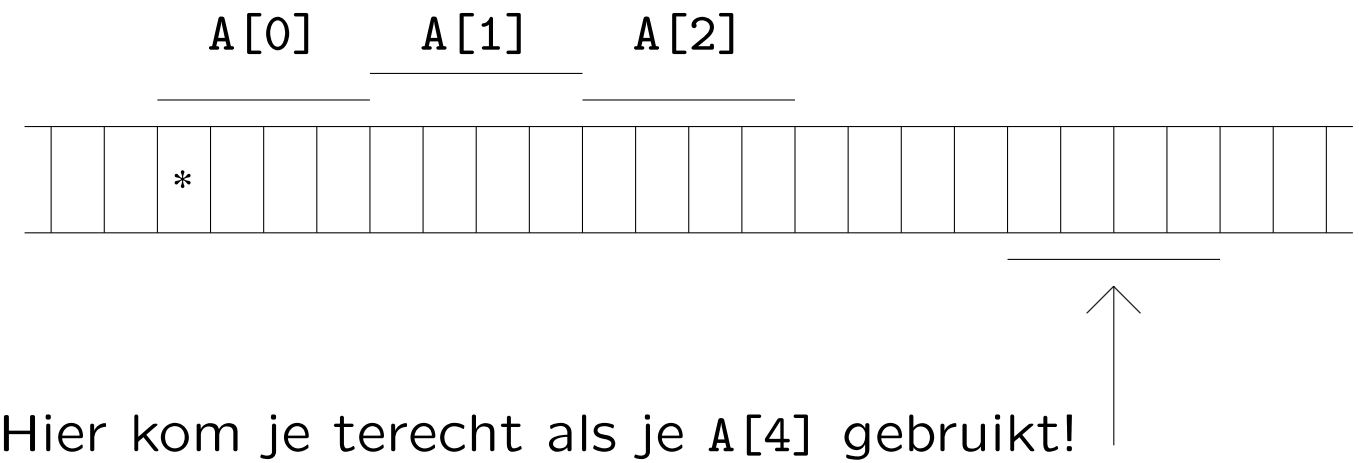
Daarna definiëren (voorlopig hetzelfde als declareren) we een array `rij` met 100 (of preciezer `MAX`) `int`'s als volgt:

```
int rij[MAX];
```

Je mag een array meteen bij definitie initialiseren (en anders alleen element voor element):

```
double B[5] = {42, 3.14, 1e6, 0, 37};
char str[10] = "feestje";  // str[7] wordt '\0'

rij[8] = 37;
rij[2] = rij[5] + rij[9];
```

Met `int A[3];` maken we een array `A` met 3 integers: `A[0]`, `A[1]` en `A[2]`, achter elkaar in het geheugen. Stel dat een `int` 4 bytes beslaat, dan benutten we in totaal dus $3 \times 4 = 12$ bytes:



Hier kom je terecht als je `A[4]` gebruikt!

Als je `cout << A << endl;` doet krijg je de waarde van `A` te zien, en dat is het geheugenadres van de eerste byte van het eerste array-element, `A[0]`, oftewel het adres van $*$.

Met `int rij[MAX];` maken we een array `rij` met `MAX` elementen dat we bijvoorbeeld als volgt gebruiken:

```
int i;  // array-index
for ( i = 0; i < MAX; i++ ) rij[i] = 5 * i;
for ( i = 0; i < MAX - 1; i++ ) rij[i] = rij[i+1];
for ( i = MAX - 1; i > 0; i-- ) rij[i-1] = rij[i];
```

Met `MAX` gelijk aan 10 wordt `rij` achtereenvolgens:

```
 0   1   2   3   4   5   6   7   8   9   <--- array-index
 0   5  10  15  20  25  30  35  40  45   <--- array-inhoud
 5  10  15  20  25  30  35  40  45  45   <--- ...
45  45  45  45  45  45  45  45  45  45   <--- ...
```

Let er op niet het array uit te lopen!
Gebruik dus nooit, ook niet indirect, `rij[MAX]` of `rij[-42]`!

Hoe druk je de inhoud van een array af?

```
void drukaf (int A[ ], int n) {
  int i;
  for ( i = 0; i < n; i++ )
    cout << A[i]; // (*)
}//drukaf
```

Of (grapje) bij (*):

```
cout << A[i] << ( i % 10 == 9 ? '\n' : ' ');
```

met de ternaire operator `...?...:...`, een voorwaardelijke expressie.

Sommigen zetten de declaratie van `i` in de for-loop:

```
for ( int i = 0; i < n; i++ ),
```

(pas dan op met geldigheid = scope van de variabele `i`).

En het minimum van een array:

```
int minimum (const int A[ ], int n) {
  int klein = A[0], i;
  for ( i = 1; i < n; i++ )
    if ( A[i] < klein )  // kleinere gevonden
      klein = A[i];
  return klein;
}//minimum
```

Die `const` verbiedt toekenningen aan array-elementen. In de heading mag ook `const int * A` staan, of `const int A[123]`. Die 123 wordt genegeerd: het gaat erom dat je doorgeeft dat het een integer-array is (de eerste parameter), met `n` elementen (de tweede parameter).

```
// Zoek getal in array A (n elementen). Lineair zoeken.
// Geeft index met A[index] = getal, als getal tenminste
// voorkomt; zo niet: resultaat wordt -1.
int lineairzoeken (int A[ ], int n, int getal) {
  int index = 0;
  bool gevonden = false;
  while ( ! gevonden && ( index < n ) ) {
    if ( getal == A[index] )
      gevonden = true;  // of meteen: return index;
    else
      index++;
  }//while
  if ( gevonden )        // en dan hier: return -1;
    return index;
  else
    return -1;
}//lineairzoeken
```

Hoe roep je functies met arrays als parameter aan?
Enkele voorbeelden, waarbij het array `rij` gedefinieerd is via `int rij[MAX];`:

`drukaf (rij,8);` (eerste 8 elementen afdrukken)

```
cout << minimum (rij,10) << endl;
```
(druk kleinste van eerste 10 elementen af)

`sorteermethode (rij,MAX);` (sorteer hele array)

`wissel (rij[5],x);` (wissel wat)

Dus *nooit* `drukaf (rij[ ],8);`!

Derde programmeeropgave — 1

Programmeermethoden

Home Algemeen Materiaal Opdrachten en Cijfers

Opdrachten en Cijfers

Programmeermethoden 2021
Derde programmeeropgave: Life

De *derde* programmeeropgave van het vak **Programmeermethoden** in het najaar van 2021 heet *Life*; zie ook het **zevende werkcollege**, en lees geregeld deze pagina op WWW.

De opgave

Het is de bedoeling om een C++-programma te maken dat de gebruiker in staat stelt *Life* te spelen via een menu-systeem. Dat betekent dat de gebruiker van het programma kan kiezen uit een aantal mogelijkheden, de zogeheten *opties*. Er is één submenu, waarin ook weer enkele opties zijn. De bedoeling is dat het hele menu op één regel staat, onder de wereld (zie verderop).
De opties worden gekozen door de eerste letter van de betreffende optie in te toetsen (gevolgd door Enter), bijvoorbeeld een `s` of `S` om te stoppen. Uiteraard wordt een en ander duidelijk en ondubbelzinnig aan de gebruiker meegedeeld. Gebruik *geen* recursie!

Alle door de gebruiker ingetoetste symbolen moeten gecontroleerd worden, dat wil zeggen dat er binnen redelijke grenzen geen foute invoer geaccepteerd wordt. Zo zal het intoetsen van bijvoorbeeld `q` of `&` in het hoofdmenu genegeerd worden. Verder moet bij getalleninvoer karakter voor karakter ingelezen worden (met `cin.get ( );` als je elders ook nog `cin >> ...` gebruikt krijg je overigens soms problemen met "hangende Enter's"; gebruik dus overal `cin.get ( )`).
Er moet ook op gelet worden dat er geen te grote getallen worden ingevoerd. Schrijf dus een geschikte functie `leesgetal` die de gelezen karakters (cijfers) omzet in een getal (tip: negeer alle "voorloop-Enter's"; verwerk alles tot en met de eerstvolgende enter, en maak hiervan zo goed mogelijk een getal, van een maximale grootte; zo kan `abc123defg999h`, als je een getal kleiner dan 10000 wilt, bijvoorbeeld verwerkt worden tot `1239`), en een functie `leesoptie` die netjes één karakter inleest en Enter's afhandelt! Aan de gebruiker mogen "redelijke" beperkingen worden gevraagd, bijvoorbeeld dat de in te voeren getallen maximaal vier cijfers hebben. Het programma moet dan echter wel bestand zijn tegen pogingen meer dan vier cijfers in te voeren. Ook het invoeren van letters in plaats van cijfers moet geen problemen opleveren. Houd het simpel!

Life is een cellulaire automaat, in 1970 bedacht door John Horton Conway. Zie verder het **college**, **Wikipedia** of **hier**, en **Johan Bontes' implementatie** [tarball van **GitHub-versie**, zie "Binary"; **patterns**]. In een 2-dimensionaal (zeg) 1000 bij 1000 rooster, de *wereld*, beginnen we met een eindig aantal levende vakjes oftewel cellen. Een levend vakje met minder dan 2 of meer dan 3 buren van de 8 (horizontaal, verticaal en diagonaal) gaat dood (uit eenzaamheid of juist overbevolking), met precies 2 of 3 levende buren overleeft het. In een dood vakje met precies 3 levende buren ontstaat leven. Dit leidt tot de volgende *generatie*. Let erop dat dit voor alle vakjes tegelijk gebeurt!
Eigenlijk moet het geheel zich afspelen op een oneindig rooster, maar we kiezen voor de eindige variant. Om moeilijkheden te voorkomen, spreken we af dat de rand van onze wereld altijd uit dode cellen blijft bestaan. De gebruiker ziet altijd een klein gedeelte van de wereld, de *view* geheten. Steeds staan de coordinaten van het punt linksboven genoemd. De hoogte en breedte van de view zijn member-variabelen (zie verderop), zeg 25 en 80. Liefhebbers mogen ze eventueel wijzigen in het parameter-submenu.

In het hoofdmenu zijn de volgende opties aanwezig:

1. Stoppen.
2. Heelschoon. Maak de wereld leeg (alle cellen gaan dood).
3. Schoon. Maak de view leeg (alle cellen in de view gaan dood).
4. Verschuif de view naar links, boven, rechts, of onder.
5. Parameters. Dit leidt tot een submenu om de parameters in te stellen, zie onder.
6. Random. Vul de view met random dode en levende cellen. De rest van de wereld blijft onveranderd.
7. Toggle. Klapt levend en dood om voor de cel op de "cursorpositie"; deze laatste kan met vier toetsen omhoog/omlaag/naar links/rechts (bijvoorbeeld W/A/S/Z) gewijzigd worden, waarbij de coordinaten

steeds getoond worden.
8. Glidergun. Vul de view met een **glidergun**. Lees de configuratie in uit een file. (Als dit "hard-coded" wordt gedaan kost dat een halve punt.)
9. Een. Er wordt één generatie gedaan.
10. Gaan. Er worden een hele serie generaties gedaan — en allemaal getoond (zonder Enter's).

Steeds wordt de view getoond, in het begin ruwweg het midden van de wereld. Voor de optie Random moet een zelfgemaakte random-generator (nou ja, random) gebruikt worden, zie Hoofdstuk 3.9.3 uit het dictaat, **gedeelte "aantekeningen bij de hoorcolleges"**.

Er zijn verschillende parameters, in te stellen via het gelijknamige submenu:

1. De *verschuivings-stapgrootte* van de view. Deze parameter wordt gebruikt als de view in één van de vier richtingen verschuift. Beeld de echte rand van de wereld ook duidelijk af, zodra deze in beeld is.
2. Het *percentage* cellen dat levend moet zijn bij de optie Random (bij benadering).
3. De *twee verschillende karakters* die op het scherm gebruikt worden voor levende en dode cellen.

Kies zelf redelijke grenswaarden voor deze parameters. En denk natuurlijk aan de optie "Terug naar het hoofdmenu".

De bedoeling is een klasse (`class`) `life` te maken, met daarin onder meer functies die ieder voor zich een menuoptie afhandelen. De parameters zijn typisch membervariabelen. Gebruik nog geen eigen headerfiles, alles moet deze keer in één file staan.

Opmerkingen
Gebruik geschikte (member)functies. Bij deze opgave mogen bij elke functie (zelfs `main`) tussen begin-{ en eind-} *hooguit circa 30* niet al te volle regels staan! Elke functie dient van commentaar voorzien te zijn, bij voorkeur één regel boven de functie. Let op goed parametergebruik: alle parameters, met uitzondering van membervariabelen, in de heading doorgeven, en de variabele-declaraties zowel bij `main` als bij de andere functies aan het begin. De enige te gebruiken headerfiles zijn in principe `iostream`, `fstream`, `cstdlib` en `string`. Zeer ruwe indicatie voor de lengte van het C++-programma: 500 regels. Denk aan het infoblokje.

Uiterste inleverdatum: **maandag 15 november 2021, 17:00 uur**.

Manier van inleveren:

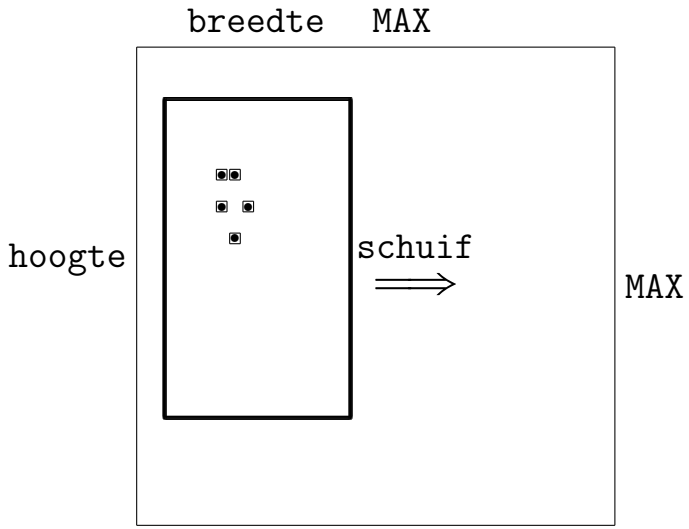
1. Digitaal de C++-code **inleveren**: stuur een email naar `pm@liacs.leidenuniv.nl`.
   Stuur geen executable's, lever alleen de C++-file digitaal in! Noem deze bij voorkeur zoiets als `jansentilanus3.cc`, dit voor de derde opdracht van het duo Jansen-Tilanus. De laatst voor de deadline ingeleverde versie wordt nagekeken.
2. En *ook* een papieren versie van het verslag (inclusief de C++-code) deponeren in de speciaal daarvoor bestemde doos "Programmeermethoden" bij kamer 159 van het Snellius-gebouw.
   Overal duidelijk datum en namen van de (maximaal twee) makers vermelden, in het bijzonder als commentaar in de eerste regels van de C++-code.
   Het *verslag* (uiteraard weer in LaTeX, zie de eerdere opgaven) moet het volgende bevatten: een beschrijving van het programma (waarin Life kort maar duidelijk wordt uitgelegd), een interessante Life-configuratie (bijvoorbeeld van internet; uiteraard met een nette citatie = referentie (`"\cite"`)) met een plaatje van een niet al te zwart screenshot (in Linux: Shift-PrintScreen) van het eigen programma waarin deze configuratie in beeld is, een beschrijving van punten waarop het programma faalt (indien van toepassing), en een tabel met gewerkte uren, uitgesplitst per week en per persoon. Geef ook minstens één andere referentie, bijvoorbeeld naar werk van Conway. Zie **hier** hoe je een plaatje verwerkt, en hoe een citatie = referentie gemaakt wordt (en **zo** ziet dat eruit).

Te gebruiken compiler: als hij maar C++ vertaalt; het programma moet in principe zowel op een Linux-machine (met `g++`) als onder Code::Blocks draaien. Test dus in principe op beide systemen! Het programma wordt doorgaans nagekeken met behulp van de compiler die (uiteraard) in het commentaar bovenin het programma vermeld staat. Normering: layout 1; commentaar (inclusief verslag) 2; modulariteit (OOP, functies) 3; werking 4. Eventuele aanvullingen en verbeteringen: lees deze WWW-bladzijde:
`www.liacs.leidenuniv.nl/~kosterswa/pm/op3pm.php`.

www.liacs.leidenuniv.nl/~kosterswa/pm/op3pm.php

Een klasse `life` voor Life ziet er $\pm$ zo uit:

```
class life {
  public:
    life ( );  // constructor
    void drukaf ( );
    void vulrandom ( );
    void maakschoon ( );
    void zetpercentage ( );
    // ...
  private:
    bool dewereld[MAX][MAX]; // array!!!
    bool reserve[MAX][MAX];  // en nog een
    int hoogte, breedte;
    int percentage, schuif;
    // ...
};//life (let op de punt-komma hier)
```



klasse object

```
life L;

L.drukaf ( );
```

Maak member-functies als (zie verderop):

```
//laat deel van de wereld zien
void life::drukaf ( );
  ...
}//life::drukaf
```

en

```
//stel percentage in tussen 0 en 100
void life::zetpercentage ( ) {
  percentage = leesGetal (100);
}//life::zetpercentage
```

waarbij de zelfgemaakte functie `int leesGetal (int maxi)` een geheel getal, maximaal `maxi`, van toetsenbord inleest.

Voor een Life-wereld is een 2-dimensionaal array nodig:

```
bool dewereld[MAX][MAX];
```

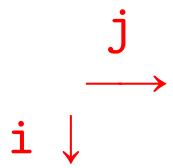
Er geldt: `dewereld[i][j]` is `true` precies dan als rij `i` (van boven) en kolom `j` (van links) een levende cel ("pixel") bevat.
En dit allemaal in een klasse `life`, met methoden als `void life::drukaf ( )`, zie eerder.

Maak eerst een *menu* en de functie `leesGetal`. Zie de tips:

www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc7.php

Een basisfunctie is dus het afdrukken van een Life-wereld:

```
//laat deel van de wereld zien; eerste poging
//te doen: de verschuifbare "view" afbeelden
void life::drukaf ( );
  int i, j;
  for ( i = 0; i < hoogte; i++ ) {
    for ( j = 0; j < breedte; j++ ) {
      if ( dewereld[i][j] )
        cout << " X";
      else
        cout << " O";
    }//for j
    cout << endl;
  }//for i
}//life::drukaf
```



- werk aan de derde programmeeropgave (menu!) — de deadline is op maandag 15 november 2021, 17:00 uur
  www.liacs.leidenuniv.nl/~kosterswa/pm/op3pm.php

- lees Savitch Hoofdstuk 5, 6 en 7.1

- lees dictaat Hoofdstuk 3.8 en 3.11

- maak opgaven 26/30 uit het opgavendictaat

- volgende week geen "reguliere" actviteiten bij Programmeermethoden

Programmeermethoden

Universiteit Leiden
The Netherlands

Arrays (vervolg)

Walter Kosters en Jonathan Vis

week 8: 1–5 november 2021

www.liacs.leidenuniv.nl/~kosterswa/pm/

week	onderwerp	boek	dictaat
1–5 nov	Arrays (vervolg)	5	4.2,op31/36
8–12 nov	Arrays (vervolg 2)	5	4.2,op37/43
15–19 nov	Pointers	10	3.12,op44/46,52/55
22–26 nov	Recursie	13	3.10,op57/61
29 nov–3 dec	Datastructuren	17	5,op47/51
6–10 dec	Algoritmen		
13–17 dec	Python, . . .		

op = opgaven uit opgavendictaat; zelf maken, antwoorden: zie website.

In rood: de weken met een deadline op de maandag erna.

Tentamen: woensdag 5 januari 2022, 14:15–17:15 uur.

Derde programmeeropgave — 1

Programmeermethoden 2021
Derde programmeeropgave: Life

De *derde* programmeeropgave van het vak **Programmeermethoden** in het najaar van 2021 heet *Life*; zie ook het **zevende werkcollege**, en lees geregeld deze pagina op WWW.

De opgave

Het is de bedoeling om een C++-programma te maken dat de gebruiker in staat stelt *Life* te spelen via een menu-systeem. Dat betekent dat de gebruiker van het programma kan kiezen uit een aantal mogelijkheden, de zogeheten *opties*. Er is één submenu, waarin ook weer enkele opties zijn. De bedoeling is dat het hele menu op één regel staat, onder de wereld (zie verderop).
De opties worden gekozen door de eerste letter van de betreffende optie in te toetsen (gevolgd door Enter), bijvoorbeeld een `s` of `S` om te stoppen. Uiteraard wordt een en ander duidelijk en ondubbelzinnig aan de gebruiker meegedeeld. Gebruik *geen* recursie!

Alle door de gebruiker ingetoetste symbolen moeten gecontroleerd worden, dat wil zeggen dat er binnen redelijke grenzen geen foute invoer geaccepteerd wordt. Zo zal het intoetsen van bijvoorbeeld `q` of `&` in het hoofdmenu genegeerd worden. Verder moet bij getalleninvoer karakter voor karakter ingelezen worden (met `cin.get ( );` als je elders ook nog `cin >> ...` gebruikt krijg je overigens soms problemen met "hangende Enter's"; gebruik dus overal `cin.get ( )`).
Er moet ook op gelet worden dat er geen te grote getallen worden ingevoerd. Schrijf dus een geschikte functie `leesgetal` die de gelezen karakters (cijfers) omzet in een getal (tip: negeer alle "voorloop-Enter's"; verwerk alles tot en met de eerstvolgende enter, en maak hiervan zo goed mogelijk een getal, van een maximale grootte; zo kan `abc123defg999h`, als je een getal kleiner dan 10000 wilt, bijvoorbeeld verwerkt worden tot `1239`), en een functie `leesoptie` die netjes één karakter inleest en Enter's afhandelt! Aan de gebruiker mogen "redelijke" beperkingen worden gevraagd, bijvoorbeeld dat de in te voeren getallen maximaal vier cijfers hebben. Het programma moet dan echter wel bestand zijn tegen pogingen meer dan vier cijfers in te voeren. Ook het invoeren van letters in plaats van cijfers moet geen problemen opleveren. Houd het simpel!

Life is een cellulaire automaat, in 1970 bedacht door John Horton Conway. Zie verder het **college**, **Wikipedia** of **hier**, en **Johan Bontes' implementatie** [tarball van **GitHub-versie**, zie "Binary"; **patterns**]. In een 2-dimensionaal (zeg) 1000 bij 1000 rooster, de *wereld*, beginnen we met een eindig aantal levende vakjes oftewel cellen. Een levend vakje met minder dan 2 of meer dan 3 buren van de 8 (horizontaal, verticaal en diagonaal) gaat dood (uit eenzaamheid of juist overbevolking), met precies 2 of 3 levende buren overleeft het. In een dood vakje met precies 3 levende buren ontstaat leven. Dit leidt tot de volgende *generatie*. Let erop dat dit voor alle vakjes tegelijk gebeurt!
Eigenlijk moet het geheel zich afspelen op een oneindig rooster, maar we kiezen voor de eindige variant. Om moeilijkheden te voorkomen, spreken we af dat de rand van onze wereld altijd uit dode cellen blijft bestaan. De gebruiker ziet altijd een klein gedeelte van de wereld, de *view* geheten. Steeds staan de coordinaten van het punt linksboven genoemd. De hoogte en breedte van de view zijn member-variabelen (zie verderop), zeg 25 en 80. Liefhebbers mogen ze eventueel wijzigen in het parameter-submenu.

In het hoofdmenu zijn de volgende opties aanwezig:

1. Stoppen.
2. Heelschoon. Maak de wereld leeg (alle cellen gaan dood).
3. Schoon. Maak de view leeg (alle cellen in de view gaan dood).
4. Verschuif de view naar links, boven, rechts, of onder.
5. Parameters. Dit leidt tot een submenu om de parameters in te stellen, zie onder.
6. Random. Vul de view met random dode en levende cellen. De rest van de wereld blijft onveranderd.
7. Toggle. Klapt levend en dood om voor de cel op de "cursorpositie"; deze laatste kan met vier toetsen omhoog/omlaag/naar links/rechts (bijvoorbeeld W/A/S/Z) gewijzigd worden, waarbij de coordinaten

steeds getoond worden.
8. Glidergun. Vul de view met een **glidergun**. Lees de configuratie in uit een file. (Als dit "hard-coded" wordt gedaan kost dat een halve punt.)
9. Een. Er wordt één generatie gedaan.
10. Gaan. Er worden een hele serie generaties gedaan — en allemaal getoond (zonder Enter's).

Steeds wordt de view getoond, in het begin ruwweg het midden van de wereld. Voor de optie Random moet een zelfgemaakte random-generator (nou ja, random) gebruikt worden, zie Hoofdstuk 3.9.3 uit het dictaat, **gedeelte "aantekeningen bij de hoorcolleges"**.

Er zijn verschillende parameters, in te stellen via het gelijknamige submenu:

1. De *verschuivings-stapgrootte* van de view. Deze parameter wordt gebruikt als de view in één van de vier richtingen verschuift. Beeld de echte rand van de wereld ook duidelijk af, zodra deze in beeld is.
2. Het *percentage* cellen dat levend moet zijn bij de optie Random (bij benadering).
3. De *twee verschillende karakters* die op het scherm gebruikt worden voor levende en dode cellen.

Kies zelf redelijke grenswaarden voor deze parameters. En denk natuurlijk aan de optie "Terug naar het hoofdmenu".

De bedoeling is een klasse (`class`) `life` te maken, met daarin onder meer functies die ieder voor zich een menuoptie afhandelen. De parameters zijn typisch membervariabelen. Gebruik nog geen eigen headerfiles, alles moet deze keer in één file staan.

Opmerkingen
Gebruik geschikte (member)functies. Bij deze opgave mogen bij elke functie (zelfs `main`) tussen begin-{ en eind-} *hooguit circa 30* niet al te volle regels staan! Elke functie dient van commentaar voorzien te zijn, bij voorkeur één regel boven de functie. Let op goed parametergebruik: alle parameters, met uitzondering van membervariabelen, in de heading doorgeven, en de variabele-declaraties zowel bij `main` als bij de andere functies aan het begin. De enige te gebruiken headerfiles zijn in principe `iostream`, `fstream`, `cstdlib` en `string`. Zeer ruwe indicatie voor de lengte van het C++-programma: 500 regels. Denk aan het infoblokje.

Uiterste inleverdatum: **maandag 15 november 2021, 17:00 uur**.

Manier van inleveren:

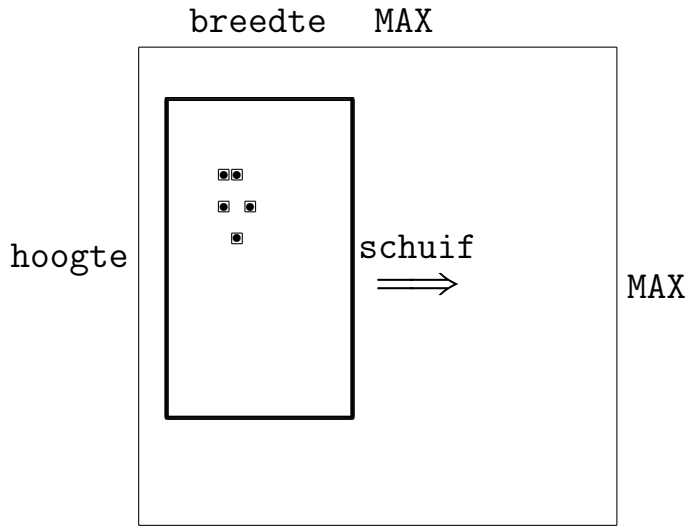
1. Digitaal de C++-code **inleveren**: stuur een email naar `pm@liacs.leidenuniv.nl`.
   Stuur geen executable's, lever alleen de C++-file digitaal in! Noem deze bij voorkeur zoiets als `jansentilanus3.cc`, dit voor de derde opdracht van het duo Jansen-Tilanus. De laatst voor de deadline ingeleverde versie wordt nagekeken.
2. En *ook* een papieren versie van het verslag (inclusief de C++-code) deponeren in de speciaal daarvoor bestemde doos "Programmeermethoden" bij kamer 159 van het Snellius-gebouw.
   Overal duidelijk datum en namen van de (maximaal twee) makers vermelden, in het bijzonder als commentaar in de eerste regels van de C++-code.
   Het *verslag* (uiteraard weer in LaTeX, zie de eerdere opgaven) moet het volgende bevatten: een beschrijving van het programma (waarin Life kort maar duidelijk wordt uitgelegd), een interessante Life-configuratie (bijvoorbeeld van internet; uiteraard met een nette citatie = referentie (`"\cite"`)) met een plaatje van een niet al te zwart screenshot (in Linux: Shift-PrintScreen) van het eigen programma waarin deze configuratie in beeld is, een beschrijving van punten waarop het programma faalt (indien van toepassing), en een tabel met gewerkte uren, uitgesplitst per week en per persoon. Geef ook minstens één andere referentie, bijvoorbeeld naar werk van Conway. Zie **hier** hoe je een plaatje verwerkt, en hoe een citatie = referentie gemaakt wordt (en **zo** ziet dat eruit).

Te gebruiken compiler: als hij maar C++ vertaalt; het programma moet in principe zowel op een Linux-machine (met `g++`) als onder Code::Blocks draaien. Test dus in principe op beide systemen! Het programma wordt doorgaans nagekeken met behulp van de compiler die (uiteraard) in het commentaar bovenin het programma vermeld staat. Normering: layout 1; commentaar (inclusief verslag) 2; modulariteit (OOP, functies) 3; werking 4. Eventuele aanvullingen en verbeteringen: lees deze WWW-bladzijde:
`www.liacs.leidenuniv.nl/~kosterswa/pm/op3pm.php`.

www.liacs.leidenuniv.nl/~kosterswa/pm/op3pm.php

Een klasse `life` voor Life ziet er $\pm$ zo uit:

```
class life {
  public:
    life ( );  // constructor
    void drukaf ( );
    void vulrandom ( );
    void maakschoon ( );
    void zetpercentage ( );
    // ...
  private:
    bool dewereld[MAX][MAX]; // array!!!
    bool reserve[MAX][MAX];  // en nog een
    int hoogte, breedte;
    int percentage, schuif;
    // ...
};//life (let op de punt-komma hier)
```

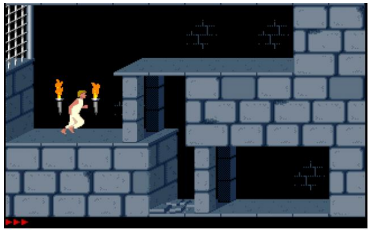


klasse object

```
life L;

L.drukaf ( );
```

Druk (een deel van) de wereld af (zie straks):

```
//laat (deel van) de wereld zien
void life::drukaf ( );
  ...
}//nono::drukaf
```



YouTube

en

```
//stel percentage in tussen 0 en 100
void life::zetpercentage ( ) {
  percentage = leesGetal (100);
}//life::zetpercentage
```

waarbij de zelfgemaakte functie `int leesGetal (int maxi)` een geheel getal, maximaal `maxi`, van toetsenbord inleest.

Voor Life is een 2-dimensionaal array nodig:

```
bool dewereld[MAX][MAX];
```

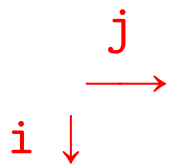
Er geldt: `dewereld[i][j]` is `true` precies dan als rij `i` (van boven) en kolom `j` (van links) een levende cel ("pixel") bevat.
En dit allemaal in een klasse `life`, met methoden als `void life::drukaf ( )`.

Maak eerst een *menu* en de functie `leesGetal`. Zie de tips:

www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc7.php

Een basisfunctie is dus het afdrukken van een Life-wereld:

```
// laat deel van de wereld zien; eerste poging
// te doen: de "verschuifbare" view afbeelden
void life::drukaf ( );
  int i, j;
  for ( i = 0; i < hoogte; i++ ) {
    for ( j = 0; j < breedte; j++ ) {
      if ( dewereld[i][j] )
        cout << " X";
      else
        cout << " O";
    }//for j
    cout << endl;
  }//for i
}//life::drukaf
```



We willen willekeurige (random) getallen maken. Dit gaat met het volgende recept, de lineaire congruentie methode (zie Knuth). Kies een startwaarde x (de "seed"). Pas dan herhaald toe

$$x = (a \cdot x + c) \text{ modulo } m$$

met vaste a, c en m; en voor modulo lees: % uit C++.

Vaak: $c = 1$, m een macht van 10, en a modulo $200 = 21$ (zie dictaat). Dan krijg je zoveel mogelijk verschillende getallen, voordat het zich gaat herhalen.

Met $x = (5 \cdot x + 1)\,\%\,8$ krijg je 0–1–6–7–4–5–2–3–0–. . .
En $x = (3 \cdot x + 1)\,\%\,8$ geeft 0–1–4–5–0–. . .

```
// geef random getal tussen 0 en 999
int randomgetal ( ) {
  static int getal = 42;                    // (*)
  getal = ( 221 * getal + 1 ) % 1000;  // niet aan knoeien
  return getal;
}//randomgetal
```

(*) Een `static` variabele is locaal, wordt eenmalig geinitialiseerd, en blijft behouden tussen functie-aanroepen.

En een random getal uit $\{1, 2, 3, 4, 5, 6\}$ aanmaken? Doe:

```
  cout << 1 + randomgetal ( ) % 6 << endl;
```

Of misschien beter `1 + randomgetal ( ) / 167` ?

(En in `cstdlib` zit `rand ( )`, en `srand (...)` zet de seed.)

Met `int A[8];` maak je een array met 8 gehele getallen: `A[0]`, `A[1]`, `A[2]`, `A[3]`, `A[4]`, `A[5]`, `A[6]`, `A[7]`.

We vullen het array:

```
A[0] = 1;
for ( int i = 1; i < 8; i++ ) A[i] = 2 * A[i-1];
```

En we drukken het af:

```
for ( int i = 0; i < 8; i++ ) cout << A[i] << " ";
```

Dat geeft `1 2 4 8 16 32 64 128 ` . Op `A[5]` staat 32.

Eendimensionale arrays

```
const int MAX = 1000;
int A[MAX]; // of int A[1000];
// declareert/definieert een array bestaande uit
// MAX integers A[0], A[1], ..., A[MAX-1]
```

en for-loop

```
for ( int i = 0; i < MAX; i++ )
  A[i] = 0;
// zet alle array-elementen op 0
```

Er is een subtiel verschil tussen *declareren* en *definiëren*.

... en functies

beginadres grootte

↓ ↓

```
void kwadraat (int B[ ], int n) { // geen & nodig!!
  for ( int i = 0; i < n; i++ )
    B[i] = i * i;
} // kwadraat

// vult de array-elementen B[0] tot en met B[n-1]
// met de eerste n kwadraten: 0 tot en met (n-1)^2
```

met aanroep: `kwadraat (A,MAX);` of `kwadraat (A,500);`. Array `A` verandert, ook al is het een call-by-value parameter!

Hoe sorteer je een array oplopend? Een eerste idee is: zet herhaald de "kleinste" vooraan.

```
void simpelsort (int C[ ], int n) {
  int voorste, kleinste, plaatskleinste, k;
  for ( voorste = 0; voorste < n; voorste++ ) {
    plaatskleinste = voorste;
    kleinste = C[voorste];
    for ( k = voorste + 1; k < n; k++ )
      if ( C[k] < kleinste ) {
        kleinste = C[k];
        plaatskleinste = k;
      }//if
    if ( plaatskleinste > voorste )
      wissel (C[plaatskleinste],C[voorste]);
  }//for
}//simpelsort
```

zoek (plaats)kleinste van C[voorste],...,C[n-1]

Een voorbeeld van de werking van simpelsort:

```
0  1  2  3  4  5  6  (n=7)

3  8  7  5  2  4  9
2| 8  7  5  3  4  9
2  3| 7  5  8  4  9
2  3  4| 5  8  7  9
2  3  4  5| 8  7  9
2  3  4  5  7| 8  9
2  3  4  5  7  8| 9
2  3  4  5  7  8  9|
```

En nog een sorteermethode:

YouTube

```
void bubblesort (int A[ ], int n) {
  int i, j;
  for ( i = 1; i < n; i++ )
    for ( j = 0; j < n - i; j++ )
      if ( A[j] > A[j+1] )
        wissel (A[j],A[j+1]);  // (*)
}//bubblesort
```

Bij (*):

```
void wissel (int & a, int & b) {
  int hulp = a; a = b; b = hulp; }//wissel
```

of (zonder functie `wissel`):

```
  { int temp = A[j]; A[j] = A[j+1]; A[j+1] = temp; }
```

Een voorbeeld van de werking van simpelsort (links) en bubblesort (rechts):

```
0  1  2  3  4  5  6  (n=7)      0  1  2  3  4  5  6

3  8  7  5  2  4  9              3  8  7  5  2  4  9
2| 8  7  5  3  4  9              3  7  5  2  4  8 |9
2  3| 7  5  8  4  9              3  5  2  4  7 |8  9
2  3  4| 5  8  7  9              3  2  4  5 |7  8  9
2  3  4  5| 8  7  9              2  3  4 |5  7  8  9
2  3  4  5  7| 8  9              2  3 |4  5  7  8  9
2  3  4  5  7  8| 9              2 |3  4  5  7  8  9
2  3  4  5  7  8  9|
```

Bubblesort doet bij een rij met n elementen

$$(n-1)+(n-2)+\ldots+3+2+1 = n(n-1)/2$$

vergelijkingen tussen array-elementen. Het is een $O(n^2)$ ("orde n^2") algoritme — en dat is niet zo fijn.

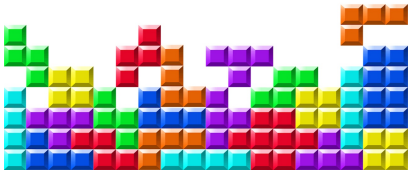
Dezelfde analyse geldt voor "simpelsort" = Selection sort.

Later meer over zoeken en sorteren . . . het kan namelijk beter = sneller!

http://www.sorting-algorithms.com/

Insertion sort (invoegsorteer)

```
// sorteer array A (met n integers) oplopend
// met behulp van insertion sort (Opgave 54)
void invoegsorteer (int A[ ], int n) {
  int i, j, temp;
  for ( i = 1; i < n; i++ ) {  // zet A[i] goed in
    temp = A[i];      // reeds gesorteerd beginstuk
    j = i - 1;
    while ( ( j >= 0 ) && ( A[j] > temp ) ) {
      A[j+1] = A[j];
      j--;
    }//while
    A[j+1] = temp;
  }//for
}//invoegsorteer
```



link

Insertion sort: werking

3	8	14	17	32	33	37	16	12	58	7	1	34	6	9

0 1 2 ... $i-1$ i ... $n-1$

okee

In de i^{de} ronde is
$A[0]$ tot en met $A[i-1]$
van array A (met n elementen)
al gesorteerd en wordt $A[i]$
in het beginstuk op de juiste
plek "ingevoegd".

Het aantal vergelijkingen tussen array-elementen dat dit sorteeralgoritme doet hangt af van het invoerrijtje (met n elementen).

In het slechtste geval (worst case) kost het

$$1 + 2 + 3 + \ldots + i - 1 + \ldots + n - 1 = \frac{1}{2}n(n-1)$$

vergelijkingen om het array oplopend te sorteren. Dit komt bijvoorbeeld voor als het beginarray aflopend gesorteerd is.

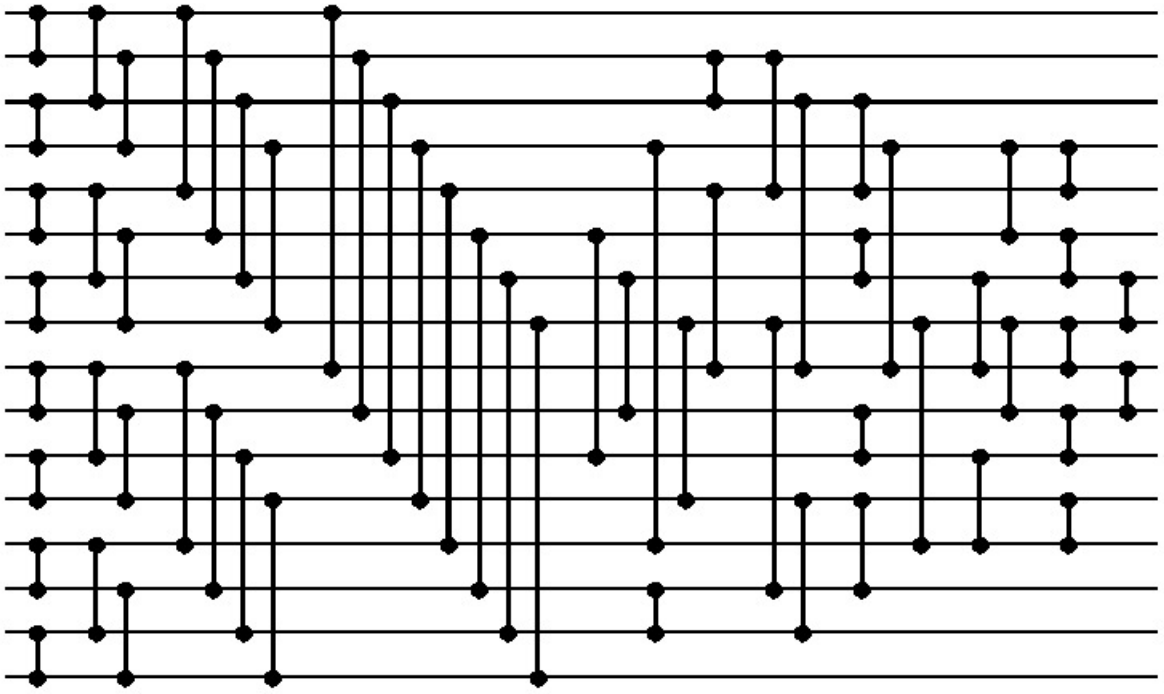
Het beste geval (best case) treedt op als het beginarray reeds oplopend gesorteerd is: $n - 1$ vergelijkingen.

Shellsort ⋈

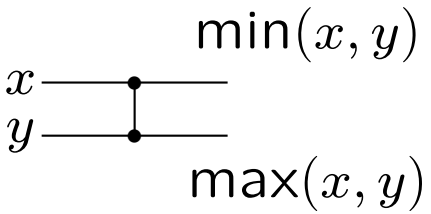
↑
geen tentamenstof

```
void Shellsort (int A[ ], int n) {
  int i, j, h = n; // spronggrootte h
  while ( h > 1 ) {
    // A is h-gesorteerd
    h = h / 2;  // er bestaan betere keuzes dan / 2
    for ( i = h; i < n; i++ ) {
      // insertion sort op deelrijtjes
      temp = A[i]; j = i - h;
      while ( ( j >= 0 ) && ( A[j] > temp ) ) {
         A[j+h] = A[j];
         j = j - h;
      }//while
      A[j+h] = temp;
    }//for
  }//while
  // h = 1: A is nu 1-gesorteerd = gesorteerd
}//Shellsort
```

Er zijn allerlei andere benaderingen, zoals “counting sort” (voor 1323312132312 3), en sorteernetwerken (GPU’s):



vergelijker:



sorteert
16 getallen
60 vergelijkers
10 tijdstappen

sorteermethode	aantal vergelijkingen (worst case)
Selection sort	$O(n^2)$
Bubblesort	$O(n^2)$
Insertion sort	$O(n^2)$
Shellsort	$O(n\sqrt{n})$ (of nog beter?)
Quicksort	$O(n \lg n)$

Deze sorteeralgoritmen zijn alle gebaseerd op het doen van array-vergelijkingen (Selection sort = simpelsort); $\lg n = {}^2\log n = \log_2 n$.

Stelling: Elk sorteeralgoritme gebaseerd op het doen van array-vergelijkingen doet in het slechtste geval (de worst case) altijd *minstens* $\lg n! \approx c\, n \lg n$ vergelijkingen voor een array met n elementen.

zie de vakken Algoritmiek en Complexiteit . . .

```
void tabelletje (int n) {
  int i, j;
  for ( i = 1; i <= n; i++ ) { // buitenste loop
    cout << i << ": ";
    for ( j = 1; j <= i; j++ ) // binnenste loop
      cout << i * j << " ";
    cout << endl;
  }//for i
}//tabelletje
```

geeft, met `tabelletje (5);`:

```
1: 1
2: 2 4
3: 3 6 9
4: 4 8 12 16
5: 5 10 15 20 25
```

Tweedimensionale arrays (2D arrays, matrices)

```
const int m = 100;     // rijen
const int n = 50;      // kolommen
int A[m][n];           // of int A[100][50];
// declareert een tweedimensionaal array van m rijen
// en n kolommen, bestaande uit m*n integers
```

en dubbele for-loop

```
int i, j;
for ( i = 0; i < m; i++ )
  for ( j = 0; j < n; j++ )
    A[i][j] = 42;      // dus niet A[i,j]
// zet alle array-elementen op 42
```

... en functies

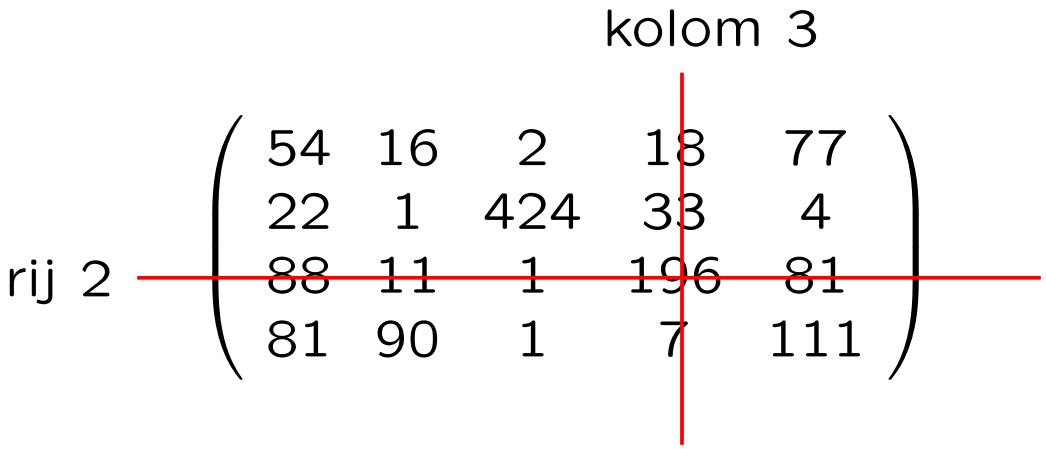
moet constante zijn!!
`B[ ][n]` mag ook (als `n const` is)

↓

```
int somarray (int B[ ][50], int zoveel) {
  int i, j, som = 0;
  for ( i = 0; i < zoveel; i++ )   // rijen
    for ( j = 0; j < 50; j++ )     // kolommen
      som += B[i][j];   // += betekent "ophogen met"
  return som;
}//somarray

// berekent de som van de elementen
// uit de eerste zoveel rijen van B
```

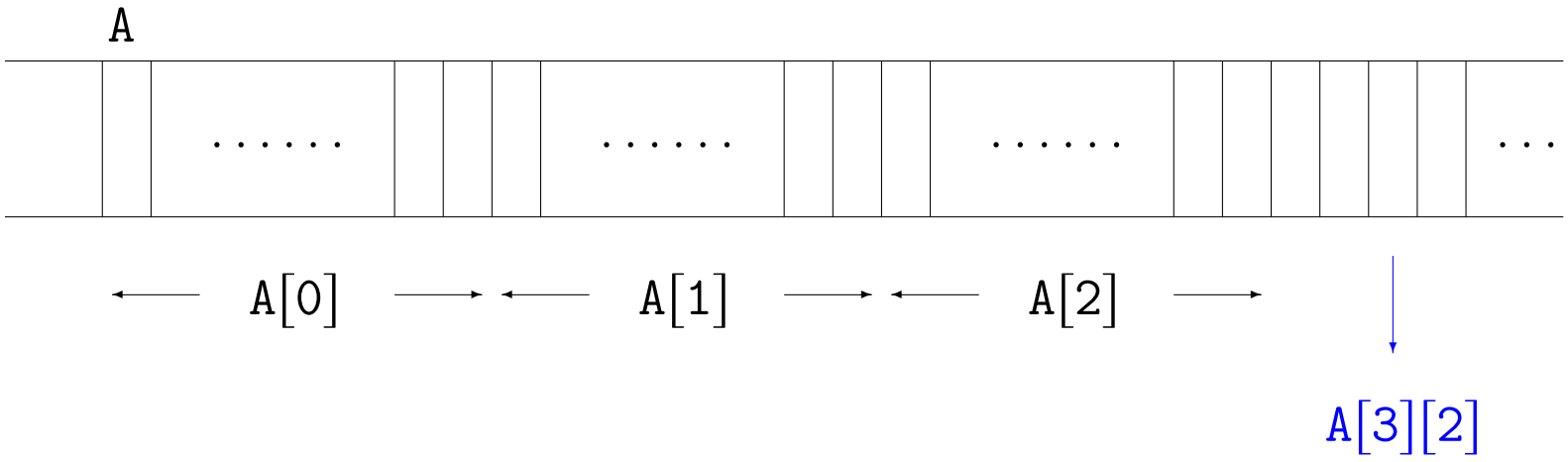
en aanroep: `antwoord = somarray (A,m);` of
`antwoord = somarray (A,20);` of ...

Een 4×5 array `A` (`int A[4][5];`) heeft 4 rijen en 5 kolommen:

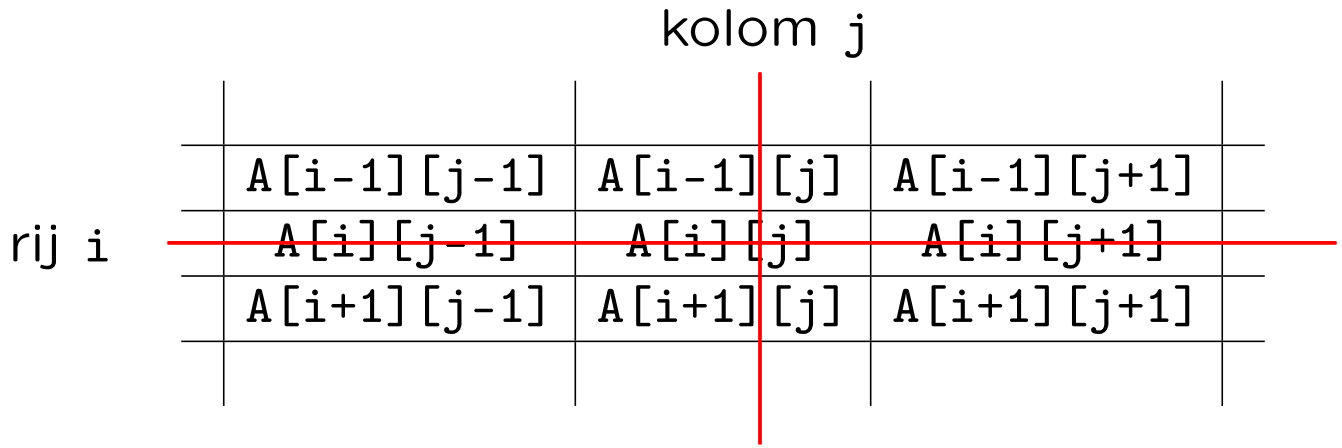


Op plek $(2, 3)$ staat het getal 196: `A[2][3]`.
Op plek $(0, 0)$ (dus linksboven) staat het getal 54: `A[0][0]`.
Rechtsonder staat `A[3][4]`: 111.
En het array-element `A[4][5]` "bestaat niet".

De rijen van zo'n 2-dimensionaal array `int A[m][n]` liggen achter elkaar in het geheugen:



Het adres van `A[3][2]` is `A + 3 * n + 2`, of eigenlijk preciezer `A + (3 * n + 2) * sizeof (int)`.
De `n` moet dus bekend, oftewel een `const`, zijn!



`m` $\times$ `n` array `int A[m][n];`

Array-elementen aan een rand hebben minder buren!

Als een functie de inhoud van het array moet veranderen is geen & nodig:

adres verandert niet ↓ dus ↓

```
void ikeerj (int A[ ][n], int m) {   // geen & !!!
  int i, j;
  for ( i = 0; i < m; i++ )
    for ( j = 0; j < n; j++ )
      A[i][j] = i * j;
}//ikeerj
```

↑ inhoud verandert wel

We bekijken nu `n` bij `n` vierkante matrices:

```
const int n = 42;

// C wordt de som van A en B, alle drie n bij n matrices
// optelling geschiedt elementsgewijs
void optellen (double A[ ][n], double B[ ][n],
               double C[ ][n]) {
  int i, j;
  for ( i = 0; i < n; i++ )
    for ( j = 0; j < n; j++ )
      C[i][j] = A[i][j] + B[i][j];
}//optellen
```

zie het vak Lineaire algebra . . .

We bekijken weer `n` bij `n` vierkante matrices:

```
const int n = 42;

// C wordt het (matrix-)product van A en B
void vermenigvuldigen (double A[ ][n], double B[ ][n],
                       double C[ ][n]) {
  int i, j, k;
  for ( i = 0; i < n; i++ )
    for ( j = 0; j < n; j++ ) {
      C[i][j] = 0;
      for ( k = 0; k < n; k++ )
        C[i][j] += A[i][k] * B[k][j];
    }//for j
}//vermenigvuldigen
```

Voorbeeld (we berekenen van $A \cdot B = C$ met name `C[1][0]`):

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

In het algemeen geldt:

$$C[i][j] = \sum_{k=0}^{n-1} A[i][k] * B[k][j] \text{ ofwel } C_{ij} = \sum_{k=0}^{n-1} A_{ik} \cdot B_{kj}$$

Het product van twee matrices is overigens ook gedefinieerd voor niet-vierkante matrices A en B, mits maar geldt dat aantal kolommen van A = aantal rijen van B.

Het gewone algoritme (links) kost $O(n^3)$ vermenigvuldigingen van array-elementen voor het product van twee $n \times n$ matrices; Strassen (rechts) "slechts" $O(n^{\log_2 7}) \approx O(n^{2.8})$:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

$$\begin{aligned} 19 &= 1*5+2*7 \\ 22 &= 1*6+2*8 \\ 43 &= 3*5+4*7 \\ 50 &= 3*6+4*8 \end{aligned}$$

$$\begin{aligned} M_1 &= (1+4)*(5+8) = 65 \\ M_2 &= (3+4)*5 = 35 \\ M_3 &= 1*(6-8) = -2 \\ M_4 &= 4*(7-5) = 8 \\ M_5 &= (1+2)*8 = 24 \\ M_6 &= (3-1)*(5+6) = 22 \\ M_7 &= (2-4)*(7+8) = -30 \\ 19 &= M_1+M_4-M_5+M_7 \\ 22 &= M_3+M_5 \\ 43 &= M_2+M_4 \\ 50 &= M_1-M_2+M_3+M_6 \end{aligned}$$

Opgave 1 van het tentamen van 6 januari 2014:

In een array `int A[n]` staan `n` (een `const` > 0) gehele getallen.

a. Schrijf een C++-functie `hoevaak (A,X,n)` die teruggeeft hoe vaak het gehele getal `X` in het array `A` voorkomt.

b. Schrijf een Booleaanse C++-functie `uniek (A,n)` die precies dan `true` teruggeeft als geen enkel getal twee maal (of vaker) voorkomt in `A`, en anders `false`. Hierbij moet de functie van **a** *zinvol* gebruikt worden (hoe vaak komt `A[i]` voor?).

c. Schrijf een C++-functie `meest (A,n)` die het meest voorkomende getal uit `A` teruggeeft. Als er verschillende kandidaten zijn (bijvoorbeeld voor het array `17 12 30 12 42 30`) moet het kleinste getal dat het meest voorkomt worden geretourneerd. In het voorbeeld is dit `12` (dat even vaak voorkomt als `30`). Maak opnieuw gebruik van de functie van **a**.

d. Schrijf een C++-functie `sorteer (A,n)` die de getallen in `A` zodanig ordent dat voor alle getallen (behalve het laatste) geldt dat ze hooguit even vaak voorkomen als hun rechter buurman. Tip: pas de C++-code voor *bubblesort* eenvoudig aan; gebruik **a**.

e. Hoe vaak wordt de functie `hoevaak` aangeroepen in **d**?

zie werkcollege 1–5 november: 1a en 2a

- werk aan de derde programmeeropgave — de deadline is op maandag 15 november 2021
- bezoek daarom de colleges, werkcolleges en vragenuren; kijk de video's
- lees Savitch Hoofdstuk 5
- lees collegedictaat Hoofdstuk 3.8, 4.1 en 4.2
- maak opgaven 31/36 uit het opgavendictaat
- www.liacs.leidenuniv.nl/~kosterswa/pm/

Programmeermethoden

Arrays (vervolg 2)

Walter Kosters en Jonathan Vis

week 9: 8–12 november 2021

www.liacs.leidenuniv.nl/~kosterswa/pm/

Voor het verslag:

- Life: . . . uitleg . . .

- citatie/referentie: `Tja~\cite{abc}` levert "Tja [1]", met

```
\begin{thebibliography}{XX}
\bibitem{abc} P.~Puk, Kabouters in de Tweede
Kamer, Ons Tijdschrift 42 (2021) 12--34.
\end{thebibliography}
```

- plaatje: `\includegraphics[height=4cm]{iets}`; let op het type (`.png`, `.jpg`, `.pdf`; gebruik zo nodig `convert`); bovenaan moet `\usepackage{graphicx}` staan; en gebruik Shift-PrtScn; zie mooier.tex

Driehoek van Pascal

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
```

↓

```
1   0
1   1   0                       A[i-1][j-1] + A[i-1][j]
1   2   1   0                               = A[i][j]
1   3   3   1   0
1   4   6   4   1   0
1   5   10  10  5   1   0
```

De getallen uit de driehoek van Pascal zijn de zogeheten binomiaalcoëfficiënten: $A[i][j] = \binom{i}{j}$.

```
void pascaldriehoek ( ) {
  int i, j;
  int Pascal[n][n];  // n een constante
  for ( i = 0; i < n; i++ )
    Pascal[i][0] = 1;  // de nulde kolom bevat enen
  Pascal[0][1] = 0;
  for ( i = 1; i < n; i++ ) {
    cout << endl << Pascal[i][0] << "  ";
    for ( j = 1; j <= i ; j++ ) {
       Pascal[i][j] = Pascal[i-1][j-1] + Pascal[i-1][j];
       cout << Pascal[i][j] << "  ";
    }//for j
    if ( i != n - 1 )
      Pascal[i][i+1] = 0;
  }//for i
}//pascaldriehoek
```

In `pascaldriehoek ( )` werd de *hele* driehoek (tijdelijk) opgeslagen in het 2-dimensionale array `Pascal`: dit kan efficiënter.

We kunnen volstaan met een 1-dimensionaal array `rij`, dat telkens de `i`-de rij uit de driehoek van Pascal bevat. Om de `(i+1)`-de rij te vinden gebruiken we immers alleen de `i`-de rij, dus alle vorige rijen zijn niet meer nodig.

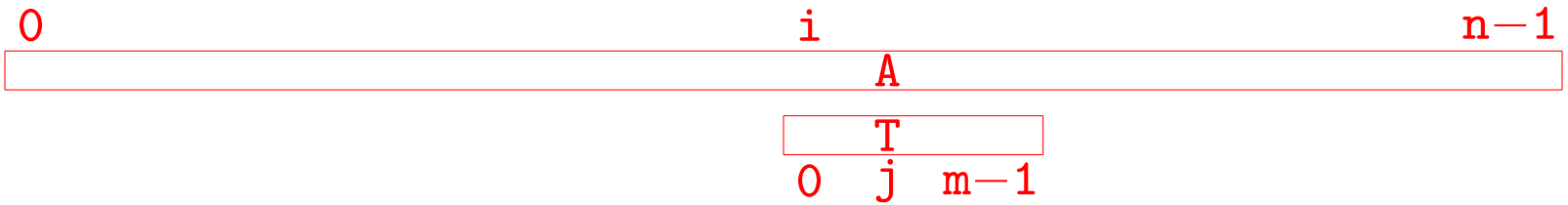
Merk op dat een volgende rij nu altijd *van rechts naar links* gevuld moet worden, omdat je anders waarden overschrijft die je nog nodig hebt.

```
void pascaldriehoekbeter ( ) {
  int rij[n];   // 1-dimensionaal array !!
  int i, j;
  for ( j = 1; j < n; j++ )  // initialisatie
    rij[j] = 0;
  rij[0] = 1;   // 0-de kolom altijd 1
  for ( i = 1; i < n; i++ ) {   // i-de rij
    for ( j = i; j > 0; j-- ) {
      rij[j] = rij[j-1] + rij[j];   // j-de element
      cout << rij[j] << "  ";
    }//for j
    cout << rij[0] << endl;  // een 1 erachter
  }//for i
}//pascaldriehoekbeter
```

Opgave 42

Komt het `m`-letter woord `woord` voor in het `n`-letter verhaal `verhaal`? Retourneer "begin" van de eerste match, of -1.

```
int komtvoor (char woord[ ], char verhaal[ ], int m, int n) {
  int i = 0, // om door verhaal heen te lopen
      j;     // om door woord heen te lopen
  bool gevonden = false;
  while ( i + m <= n ) {
    gevonden = true;                      // optimist
    for ( j = 0; j < m; j++ )             // bot
      if ( woord[j] != verhaal[i+j] ) // pech
        gevonden = false;
    if ( gevonden )                       // bingo
      return i;
    i++;
  }//while
  return -1;
}//komtvoor
```



Er zijn talloze patroonherkennings-algoritmen die sneller een (korte) string in een (lange) string opsporen.
Voorbeelden zijn het Boyer-Moore algoritme en het Knuth-Morris-Pratt algoritme, zie het college Datastructuren.

Stel je zoekt `BABBM`, en ziet de mismatch `A`↔`M`:

```
U V W B A B B A T L K ...
      B A B B M
```

Je kunt dan doorschuiven naar

```
U V W B A B B A T L K ...
            B A B B M
```

en `T` met `B` gaan vergelijken.

Donald "TEX" Knuth

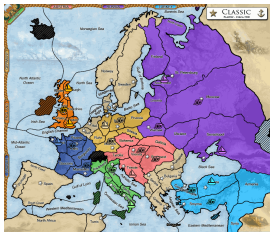
Gevraagd: rij uit `bord` die geheel uit klinkers bestaat

```
int rijklinkers (const char bord[8][8]) {
  int i = 0, j = 0, rij = -1;
  bool okee;
  for ( i = 0; i < 8; i++ ) {
    okee = true;   // allemaal klinkers in rij i?
    for ( j = 0; j < 8; j++ ) {
      if ( ! ( ( bord[i][j] == 'A' ) || ( bord[i][j] == 'E' )
               || ( bord[i][j] == 'I' ) || ( bord[i][j] == 'O' )
               || ( bord[i][j] == 'U' ) ) )
        okee = false;   // bord[i][j] is geen klinker
    }//for j
    if ( okee )
      rij = i;
  }//for i
  return rij;   // -1 als zo'n rij niet bestaat
}//rijklinkers
```

Alle 64 array-elementen worden bekeken, 't kan beter . . .

Het kan zelfs *efficiënter* en mooier met *één while-loop*:

```
int klinkerrij (const char bord[ ][8]) {
  int i = 0, j = 0, rij = -1;
  bool stoppen = false;
  while ( ! stoppen && ( i < 8 ) )
    if ( ( bord[i][j] == 'A' ) || ( bord[i][j] == 'E' )
         || ( bord[i][j] == 'I' ) || ( bord[i][j] == 'O' )
         || ( bord[i][j] == 'U' ) )
      if ( j == 7 ) {  // bingo
        stoppen = true;
        rij = i;  // of: return i;
      }//if
      else
        j++;
    else {  // volgende rij vooraan
      i++;
      j = 0;
    }//else
  return rij;   // -1 als zo'n rij niet bestaat
}//klinkerrij
```



Opgave 43d: plaatje

	0	1	2	3	4	5	6	7
0	E	E	O	T				
1	I	X						
2	A	I	O	A	A	A	Z	
3	U	X						
4	R							
5	O	I	O	I	A	A	U	A
6								
7								

Vraag: begint op positie (i,j) in puzzel (met # als zwart vakje) een nieuw horizontaal woord?

```
bool bestaatHoriWoord (char puzzel[ ][n], int i, int j) {
  if ( ( puzzel[i][j] != '#' ) &&
       ( ( j == 0 ) || ( puzzel[i][j-1] == '#' ) ) &&
       ( ( j != n-1 ) && ( puzzel[i][j+1] != '#' ) ) )
    return true;
  else                          //   H E T #
    return false;               //   A # O M
}//bestaatHoriWoord            //   # S P A
```

Er wordt rijkelijk gebruik gemaakt van C++-short-circuiting evaluatie.

Eén statement `return ( ( puzzel[i][j] != ...` kan ook.

Opdracht: vul het array `nummers` (rij voor rij en per rij van links naar rechts): vakjes waar een woord begint (horizontaal of verticaal) krijgen een rangnummer, andere 0:

```
void nummeren (char puzzel[ ][n], int m,
               int nummers[ ][n]) {
  int i, j; int teller = 1;   // rangnummer
  for ( i = 0; i < m; i++ )
    for ( j = 0; j < n; j++ )
      if ( bestaatHoriWoord ( puzzel, i, j ) ||
           bestaatVertiWoord ( puzzel, i, j ) ) {
        nummers[i][j] = teller;
        teller++;            //   H E T #   1 0 2 0
      }//if                   //   A # O M   0 0 3 4
      else                   //   # S P A   0 5 0 0
        nummers[i][j] = 0;
}//nummeren
```

Voorbeeld:

```
const int l = 500;       // 500 bladzijden
const int m = 45;        // 45 regels per bladzijde
const int n = 75;        // 75 karakters per regel
char boek[l][m][n];      // of char boek[500][45][75];
// declareert een driedimensionaal array van karakters
// het heeft 500 * 45 * 75 = 1687500 bytes
```

`boek[342][27][55]` betekent: van pagina 342, regel 27 het 55-e karakter; denk er aan dat ze allemaal bij `0` beginnen met indiceren.
(En gebruik nooit `l` als variabele! (`\ell` $\rightarrow \ell$))

Een priemgetal is een geheel getal > 1 dat alleen door 1 en zichzelf deelbaar is:

```
bool priem (int getal) {
  int deler = 2;
  double wortel = sqrt (getal); // uit <cmath>
  bool geendelers = true; // optimist!
  while ( ( deler <= wortel ) && geendelers ) {
    if ( getal % deler == 0 ) // (*)
      geendelers = false;     // (*)
    deler++;
  }//while
  return geendelers;
}//priem
```

Bij (*) mag ook staan: `geendelers = ( getal % deler != 0 );`

Dat kan ook als volgt, waarbij we gebruiken dat `return` de functie meteen stopt:

```
bool priem2 (int getal) {
  int deler = 2;
  double wortel = sqrt (getal); // uit <cmath>

  while ( deler <= wortel ) {
    if ( getal % deler == 0 )
      return false;
    deler++;
  }//while
  return true;
}//priem2
```

Er zijn veel snellere programma's!

Ontbind een getal in factoren, zoals $84 = 2^2 \times 3^1 \times 7^1$:

```
void ontbinden (int getal) {
  int teller = 0;  // hoe vaak past deler in getal?
  int deler = 2;   // kandidaat deler
  while ( getal != 1 ) {
    if ( getal % deler == 0 ) {
      teller = 0;
      while ( getal % deler == 0 ) {
        getal = getal / deler;
        teller++;
      }//while
      cout << deler << " tot de " << teller << "-de macht ";
    }//if
    deler++;
  }//while
}//ontbinden
```

De zeef van Eratosthenes vindt priemgetallen:

```
bool zeef[MAX];  // getal i priem <=> zeef[i] true
int getal, veelvoud;
double wortel = sqrt (MAX);
zeef[0] = false;
zeef[1] = false;
for ( getal = 2; getal < MAX; getal++ )
  zeef[getal] = true;  // ... tot het tegendeel bewezen is
for ( getal = 2; getal < wortel; getal++ )
  if ( zeef[getal] ) {  // streep veelvouden door
    veelvoud = 2 * getal;  // getal + getal als * "te duur" is
    while ( veelvoud < MAX ) {
      zeef[veelvoud] = false;
      veelvoud = veelvoud + getal;
    }//while
  }//if
for ( getal = 2; getal < MAX; getal++ ) {
  if ( zeef[getal] )
    cout << getal << "  ";
}//for
```

Hoe werk je in de praktijk met rijtjes `char`'s, oftewel strings? Dat kan in C++ op twee manieren:

- Met "ouderwetse" C-strings:
  ```
  char woord[7] = "Het.";
  ```
  Nu wordt `woord[4]` gelijk aan `'\0'`, en zit er troep in `woord[5]` en `woord[6]`.

- Beter: met nieuwe strings uit de string-klasse, via:
  ```
  #include <string>
  string woord = "De.";
  ```
  Nu heb je `woord.length ( )`, hier 3, array-elementen, zoals `woord[1]`, waar `'e'` in zit. En `woord.c_str ( )` levert de overeenkomende C-string. Maar hoe die ingewikkelde objecten opgeslagen zitten?

Je kunt beide types strings inlezen, maar er zijn gevaren:

- C-strings "groeien niet mee", strings uit de string-klasse wel.

- Met `cin >> regel;` lees je in de (C-)string `regel` in, maar dit stopt bij "whitespace"! Gebruik liever `getline`.

- Als je doet

```
cin >> n;  // een int
getline (cin, regel);  // een string
```

  wordt `regel` de lege string: de vorige Enter!

En tot slot: met strings uit de string-klasse werken ==, en zelfs < en <= (lexicografische ordening), met C-strings niet . . . want daar vergelijk je met == adressen.

Hoe kun je parameters doorgeven aan een programma?
Dat gaat via C-strings, mee te geven aan de functie `main`:

```
int main (int argc, char* argv[ ]) {
  if ( argc > 1 )  // telt aantal parameters
    cout << "===Parameter " << argv[1] << endl;
  cout << "===Executable heet " << argv[0] << endl;
}//main
```

Als je dit programma (zeg `iets.cc`) compileert met
`g++ -Wall -o doewat iets.cc` levert `./doewat 1234` op:

```
===Parameter 1234
===Executable heet ./doewat
```

⋈ geen tentamenstof

En wat gebeurt er als je een array kopieert met `A = B;`? Dan wijzen ze naar ("zijn ze") dezelfde inhoud, maar is het oude array `A` "zoek". Je kopieert namelijk het beginadres. Dus beter is:

```
void kopieer (int A[ ], int B[ ], int n) {
  int i;
  for ( i = 0; i < n; i++ )
    A[i] = B[i];
}//kopieer
```

En let op: geen `&` te zien.
Voor `int B[ ]` zou `const` mogen staan.

De array-grootte moet eigenlijk een constante zijn, dus:

```
int n;
cin >> n;
int A[n];  // verboden! (maar compileert meestal wel)
int B[10]; // in orde
```

Wat goed is, maar ook wat nadelen heeft, is:

```
int n;
cin >> n;
A = new int[n];  // dynamisch array (zie pointers ...)
...
delete [ ]  A;   // ook zelf opruimen!
```

⋈ geen tentamenstof

OK, en arrays met rijen van variabele lengte? En dynamische arrays?

```
int** A = new int*[m];
for ( int i = 0; i < m; i++ ) A[i] = new int[n];

... gebruik A[i][j] ...

for ( int i = 0; i < m; i++ ) delete [ ] A[i];
delete [ ] A;
```

⋈ geen tentamenstof

Opgave 3 van het tentamen van 3 januari 2018:

Gegeven is het 2-dimensionale array `int stemmen[m][n];`, met zekere `const int m` $\geqslant 2$ en `const int n` $\geqslant 2$. Er geldt dat `stemmen[i][j]` $\geqslant 0$ het aantal stemmen van jurylid `i` op liedje `j` is. Een voorbeeld met `m = 4` en `n = 5`:

```
2  5  0  3  2
1  5  1  4  1
2  1  3  3  3
8  1  0  0  3
```

a. Schrijf een Booleaanse C++-functie `eerlijk (stemmen)` die controleert of alle juryleden exact evenveel stemmen hebben uitgebracht. Oftewel: hebben alle rijen dezelfde totaalsom? In het voorbeeld: `true`.

b. Twee juryleden stemmen *ongeveer hetzelfde* als voor ieder liedje geldt dat hun aantal stemmen hooguit één verschilt, en tevens dat bij minstens één liedje er evenveel stemmen zijn. Schrijf een Booleaanse C++-functie `idem (stemmen,i1,i2)` die dit controleert voor de juryleden `i1` en `i2`. Neem aan dat $0 \leqslant$ `i1` $<$ `m` en $0 <$ `i2` $<$ `m`. In het voorbeeld `true` bij `0` en `1`.

c. Begin bij een zeker jurylid `i1`, met $0 \leqslant$ `i1` $<$ `m`. Zoek het nieuwe jurylid `i2` dat ongeveer hetzelfde stemt als `i1` (gebruik **b**), waarbij `i2` zo klein mogelijk is. Zoek vanuit `i2` het nieuwe (verschillend van `i1` en `i2`) jurylid `i3` (`i3` zo klein mogelijk) dat ongeveer hetzelfde stemt als `i2`, enzovoorts: `i4` verschilt van `i1`, `i2` en `i3`, en stemt ongeveer hetzelfde als `i3`, ... Je stopt indien je een dergelijk nieuw jurylid niet meer kunt vinden. Schrijf een C++-functie `int vrienden (stemmen,i1)` die dit doet, en teruggeeft hoeveel juryleden je zo gevonden hebt, inclusief `i1`. Tip: gebruik een Booleaans hulparray.

Opgave 3 van het tentamen van 16 maart 2017:

Gegeven is het twee-dimensionale array `int afstand[n][n];`, met zekere `const int n` $\geqslant 2$. Er geldt dat `afstand[i][j]` > 0 de afstand is tussen de plaatsen `i` en `j` met `i` $\neq$ `j`, waarbij `afstand[i][i]` `0` is, en de afstand tussen `i` en `j` even groot is als die tussen `j` en `i`. Een voorbeeld met `n = 4` staat hiernaast.

0	3	7	9
3	0	4	14
7	4	0	8
9	14	8	0

a. Schrijf een C++-functie `bool reis (afstand,km)` die kijkt of er een rondreis van `i` naar `j` naar `k` naar `i` is (voor willekeurige onderling verschillende plaatsen `i`, `j` en `k`) die in totaal precies afstand `km` heeft. In het voorbeeld zou voor `26` het antwoord `true` zijn: van `0` naar `1` naar `3` naar `0` (dat is `3 + 14 + 9` $=$ `26`).

b. Schrijf een C++-functie `int verste (afstand,i)` die het nummer van de verst van `i` afgelegen plaats oplevert. In het voorbeeld, met `i` $=$ `3`, is dat `1` (wegens afstand `14`). Als er meerdere plaatsen voldoen: die met de grootste index. Neem aan dat `0` $\leqslant$ `i` $<$ `n`.

c. Schrijf een C++-functie `int hoever (afstand,i)` die bepaalt hoeveel je in totaal reist als je begint in `i`, dan steeds naar de verst gelegen plaats gaat, en stopt zodra je ergens komt waar je al eerder geweest was. In het voorbeeld, met `i` $=$ `0`, is het antwoord `9 + 14 + 14` $=$ `37` (van `0` naar `3` naar `1` naar `3`). Neem aan dat `0` $\leqslant$ `i` $<$ `n`. Hint: gebruik een Booleaans hulparray.

Uitwerking Opgave 3c van tentamen 16 maart 2017:

```
int hoever (int afstand[ ][n], int i) {
  int afst = 0, j;
  bool geweest[n];
  for ( j = 0; j < n; j++ )
    geweest[j] = false;
  while ( ! geweest[i] ) {
    geweest[i] = true;
    j = verste (afstand,i);
    afst += afstand[i][j];
    i = j;
  }//while
  return afst;
}//hoever
```

- werk aan de derde programmeeropgave — de deadline is op maandag 15 november 2021
  bezoek daarom de werkcolleges en vragenuren
- volgende week: pointers!
- lees Savitch Hoofdstuk 5
- lees collegedictaat Hoofdstuk 3.8, 4.1 en 4.2
- maak opgaven 37/43 uit opgavendictaat
- www.liacs.leidenuniv.nl/~kosterswa/pm/

Programmeermethoden

Pointers

Walter Kosters en Jonathan Vis

week 10: 15–19 november 2021

www.liacs.leidenuniv.nl/~kosterswa/pm/

Een pointer is in feite gewoon een *geheugenadres*. Het geheugen kun je je voorstellen als een lineaire lijst met bytes.

Als `p` een pointer is naar een geheel getal `i` (gedefinieerd via `int * p = &i;`), is `p` het adres van `i`, en is `i` ook als `*p` te benaderen:

```
i = 196;   ⟺   *p = 196;
```

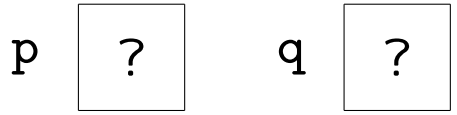
Met behulp van pointers kun je het geheugen dynamisch beheren, dat wil zeggen tijdens het runnen van het programma.

Stel je wilt jaargangen Donald Duck opbergen in een bibliotheek, in verschillende kasten, en maar één cataloguskaartje gebruiken.

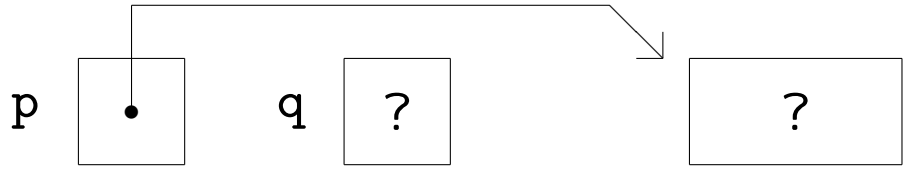
Oplossing: plak de plek van de eerstvolgende jaargang achterin de vorige. Dus achterin 2007 zit een sticker met "jaargang 2008 staat op plek . . . ", achterin 2020 staat "dit is de laatste". En de eerste is 1952.

Voorbeeld — 1

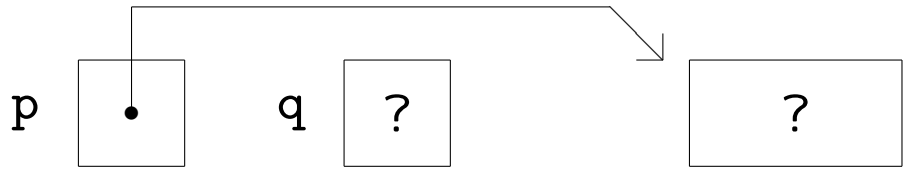
⋈ `int * p;  int * q;` — twee pointers naar `int`(eger)



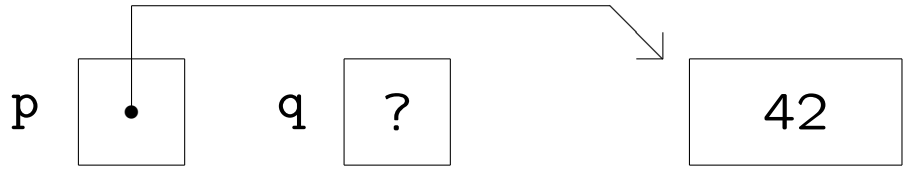
⋈ `p = new int;` — maak nieuwe `int`
en stop diens adres in `p`



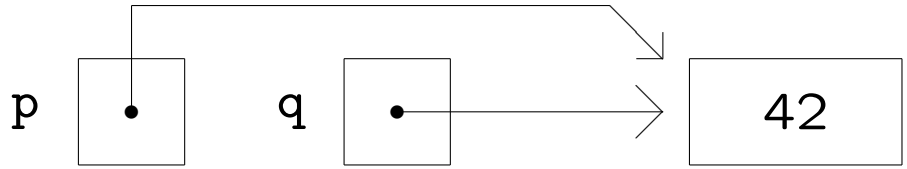
Voorbeeld — 2



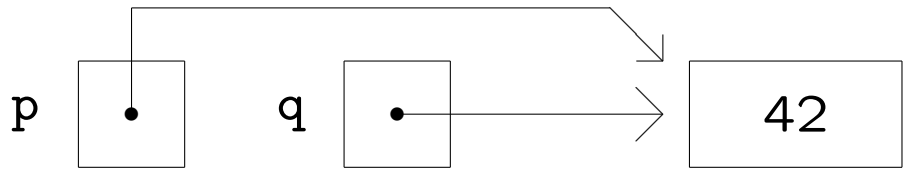
⋈ `*p = 42;` — stop 42 in die `int`



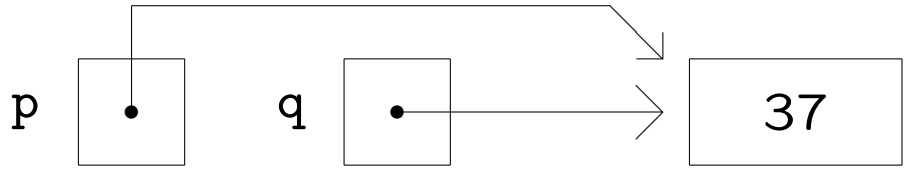
⋈ `q = p;` — laat `q` diezelfde `int` aanwijzen



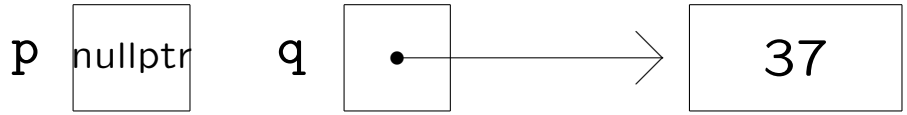
Voorbeeld — 3

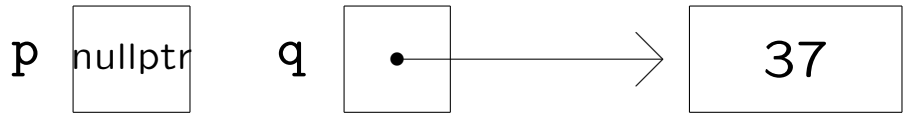


⋈ `*q = 37;` — verander de `int` via `q`

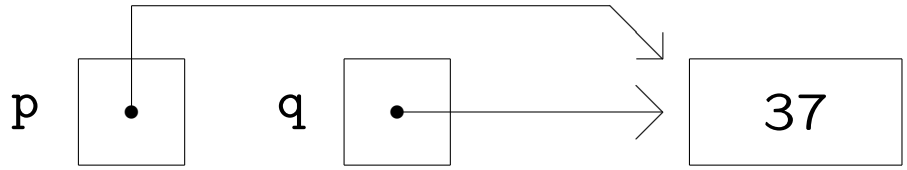


⋈ `p = nullptr;` — laat `p` naar "niets" wijzen

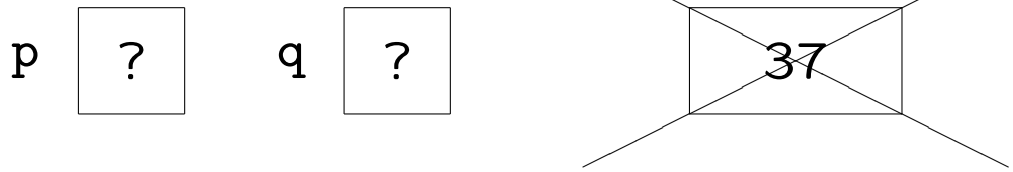




⋈ `p = q;` — zet `p` weer "terug"



⋈ `delete q;` — blaas (via `q`) de `int` op



Bij het deleten wordt de geheugenruimte weer vrijgegeven. Bij `delete q;` wordt niet `q` vrijgegeven, maar datgene waar `q` naar wijst!

Voorlopig bevat `*q` nog wel de oude waarde (37) maar op enig moment niet meer . . .

Goed: `delete q; q = nullptr;`, want nu weet je zeker dat `q` naar "niets" wijst.

Sinds 2011 gebruiken we `nullptr` in plaats van `NULL` (dat is gewoon 0); doe zonodig `g++ -std=c++11 -Wall -Wextra ...`

Als je een pointer afloopt ("dereferencen") moet je zeker weten dat deze niet de `nullptr` is!

Fout is zoiets als

```
q = new int; q = p;
```

Je maakt nu namelijk eerst een nieuwe `int`, laat `q` daarnaar wijzen, en verandert `q` dan. Het adres van die eerste `int` is nu "voor altijd" zoek, en die `int` verspilt ruimte!

Slecht is:

```
p = nullptr; delete p;
```

Een "nullptr" kun je niet deleten — als je het toch doet, gebeurt er niks.

```
int i;        // een integer
int* p;       // een pointer naar een integer
p = &i;       // p wijst i aan: p is nu het adres van i
i = 12;       // verandert i
*p = *p + 8; // oftewel *p += 8;, verandert i opnieuw
int *q = &i; // q wijst ook i aan; geen new's,
              // en zeker geen delete's!
```

Let op: `int* p, q;` betekent dat `p` een pointer naar een integer is, en `q` een integer; bij `int* p; int* q;` en ook bij `int *p, *q;` heb je twee pointers naar een integer.

En wat betekent `*p++`? Is dat `(*p)++` of `*(p++)`?

En met klassen erbij:

```
class wagon {
  public:
    int hoogte;
    ...
};//wagon

wagon *p;  // p is pointer naar (= adres van) een wagon
```

Nu kun je de hoogte van de door `p` aangewezen wagon `*p` (de member-variabele `hoogte`) benaderen via `(*p).hoogte`, maar ook via (nieuwe notatie) `p->hoogte`.

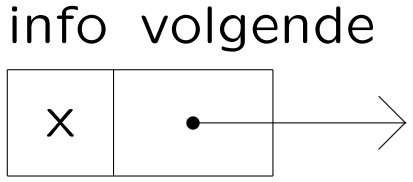
Enkelverbonden pointerlijst — 1

We maken nu een enkelverbonden pointerlijst bestaande uit vakjes:

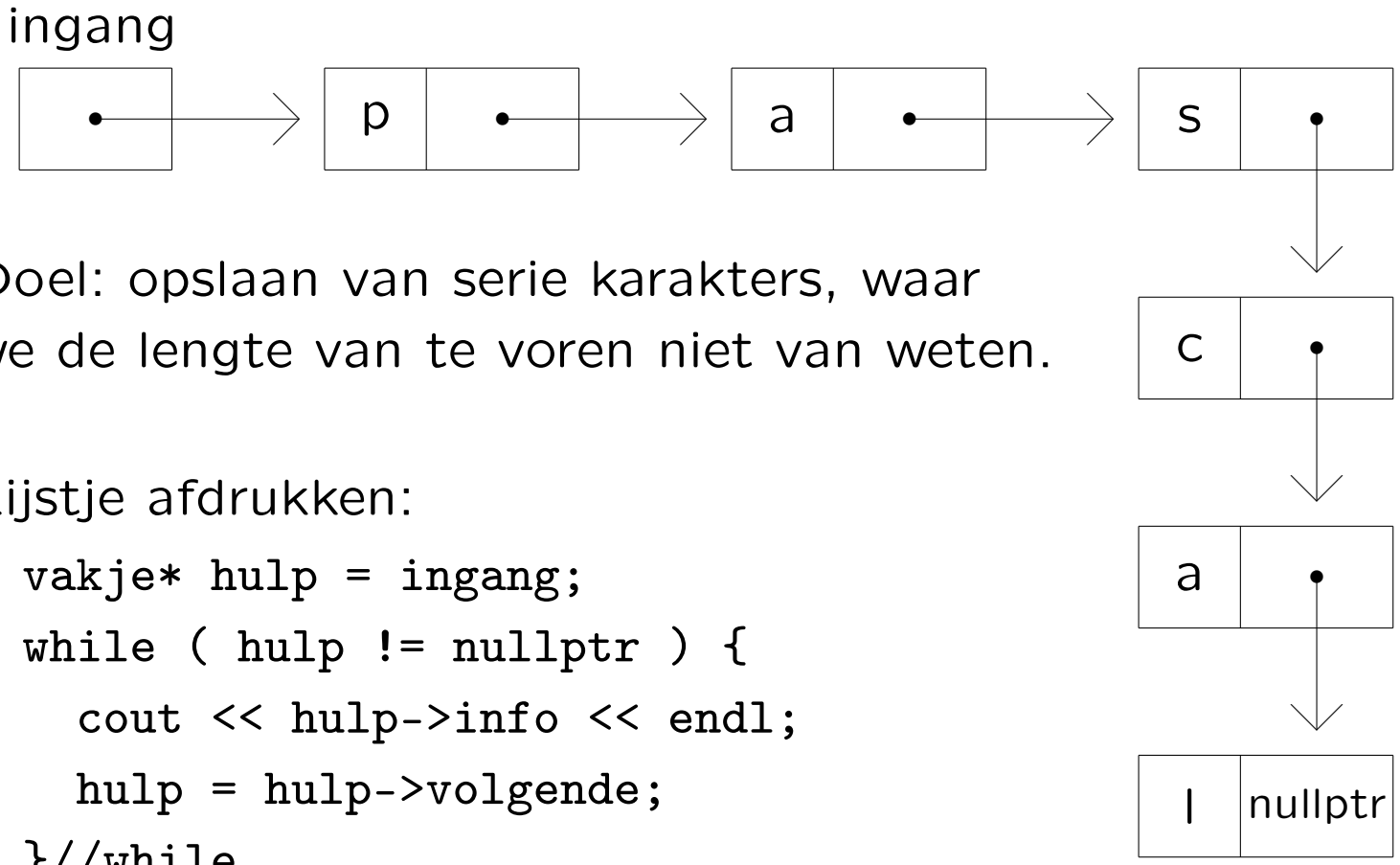
```
class vakje {
  public:
    char info;
    vakje* volgende;
};//vakje
```

Vaak wordt hiervoor in plaats van `class` het iets eenvoudigere `struct` gebruikt.

Zo'n vakje ziet er uit als:



We willen bijvoorbeeld maken:



Doel: opslaan van serie karakters, waar we de lengte van te voren niet van weten.

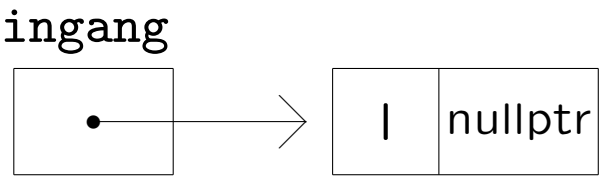
Lijstje afdrukken:

```
vakje* hulp = ingang;
while ( hulp != nullptr ) {
  cout << hulp->info << endl;
  hulp = hulp->volgende;
}//while
```

Hoe bouwen we zo'n lijst vanaf niets op?

```
vakje* ingang;
ingang = new vakje;
ingang->info = 'l';
ingang->volgende = nullptr;
```

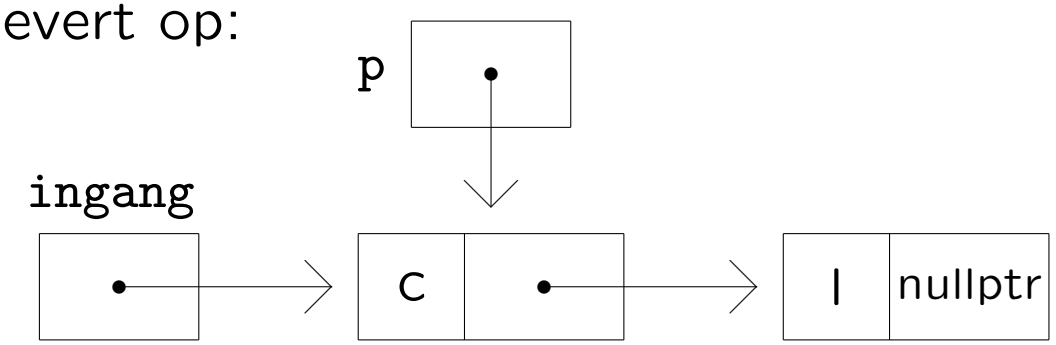
Dit levert op:



Hoe duwen we er nog een vakje voor?

```
vakje* p;
p = new vakje;
p->info = 'c';
p->volgende = ingang;
ingang = p;
```

Dit levert op:

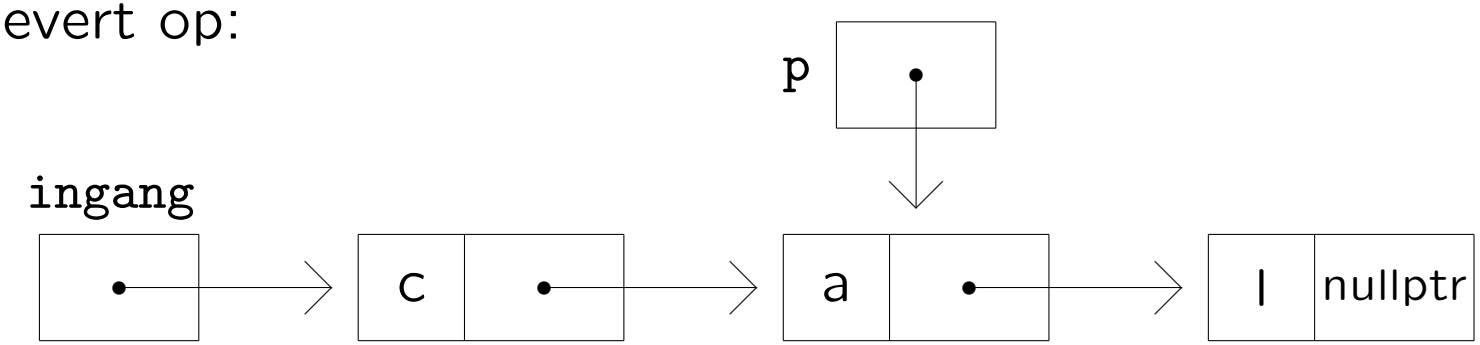


Let op: nu *niet* `delete p;` zeggen!

Hoe stoppen we er een vakje tussen?

```
vakje* p;
p = new vakje;
p->info = 'a';
p->volgende = ingang->volgende;
ingang->volgende = p;
```

Dit levert op:



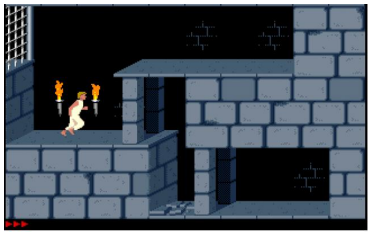
Etcetera . . .

Maar het kan uiteraard beter met een functie:

```
class vakje { public:
  char info;
  vakje* volgende;
};//vakje

void zetervoor (char letter, vakje* & ingang) {
  vakje* p;                          // let op de &
  p = new vakje;
  p->info = letter;
  p->volgende = ingang;
  ingang = p;  // en nu NIET delete p;!
}//zetervoor

ingang = nullptr; // met vakje* ingang;
zetervoor ('l',ingang); zetervoor ('a',ingang);
zetervoor ('c',ingang); zetervoor ('s',ingang);
zetervoor ('a',ingang); zetervoor ('p',ingang);
```

's Avonds lijst bewaren (zonder pointers):

```
vakje* hulp;
while ( ingang != nullptr ) {
  uitvoer.put (ingang->info);
  hulp = ingang;
  ingang = ingang->volgende;
  delete hulp;
}//while
```



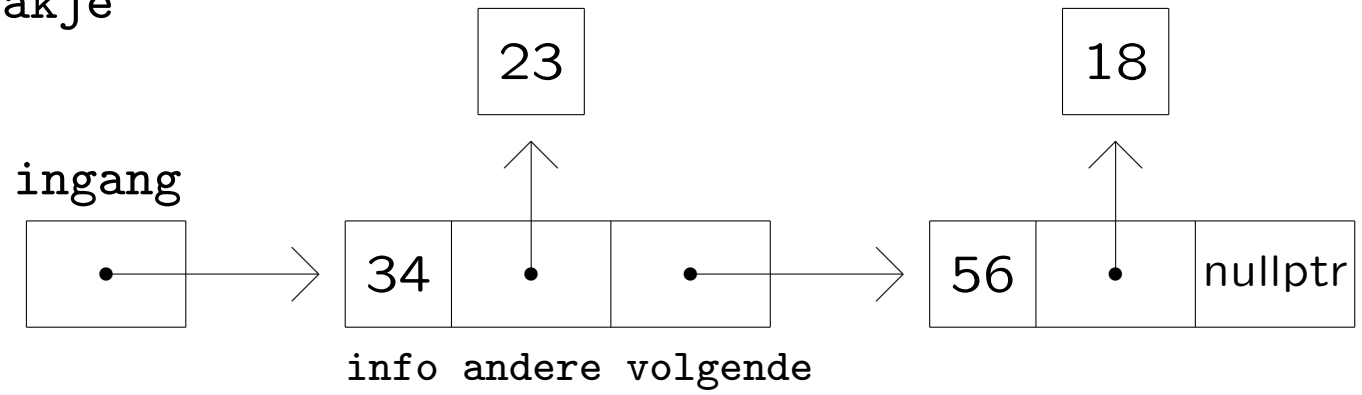
's Ochtends lijst weer opbouwen (met pointers):

```
ingang = nullptr; char letter = invoer.get ( );
while ( ! invoer.eof ( ) ) {
  zetervoor (letter,ingang);
  letter = invoer.get ( );
}//while
```

Helaas . . . verkeerd om: nog eens wegschrijven en weer opbouwen, of `zeterachter` schrijven (laatste onthouden).

In het werkcollege (18–19 november 2021) gebruiken we:

```
class vakje {
  public:
    int info;
    vakje* volgende;
    int* andere;
};//vakje
```

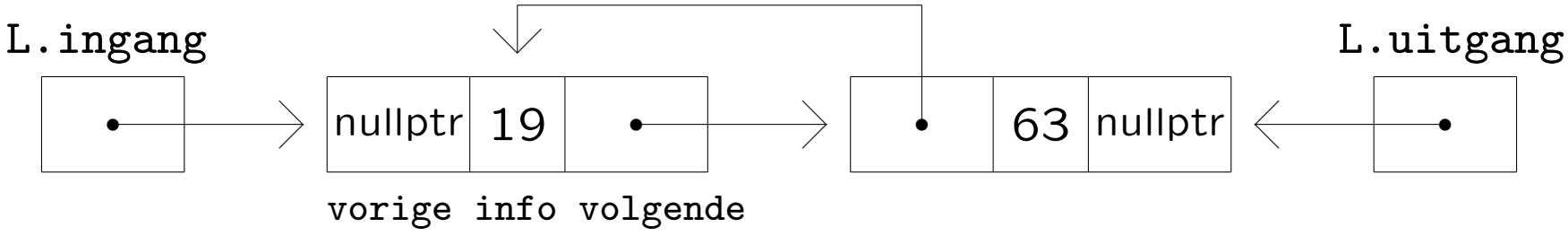


www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc10.php

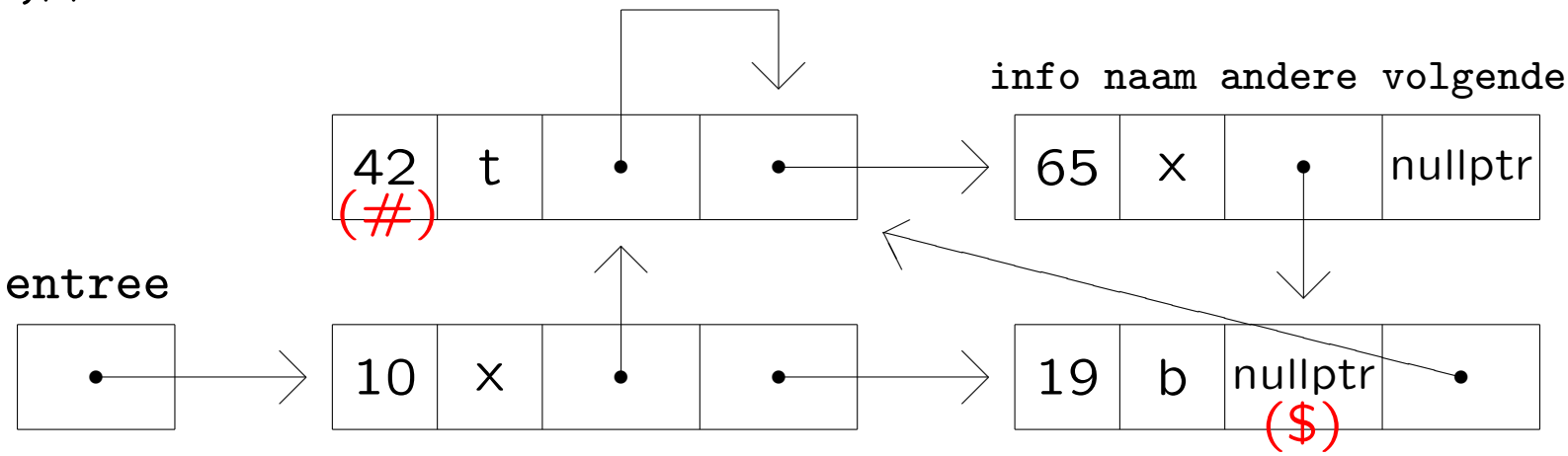
En voor een dubbel-verbonden pointerlijst:

```
class element {
  public:
    element* vorige;
    int info;
    element* volgende;
};//element

lijst L;
```

```
class lijst {
  private:
    element* ingang;
    element* uitgang;
  public:
    void afdrukkenVA ( );
    ...
};//lijst
```



```
class vanalles { public:
  int info; char naam;
  vanalles* volgende; vanalles* andere;
};//vanalles
```



($) `entree->volgende->andere`

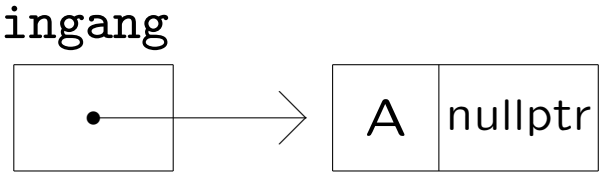
(#) `entree->volgende->volgende->info` of `entree->andere->info`

Controle op geheugenlekkages: `valgrind ./hetprogramma`

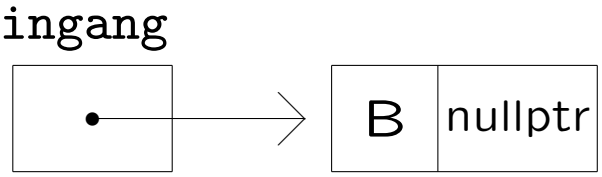
Bekijk de volgende functie:

```
void demo (char letter, vakje*   ingang) {
  if ( ingang != nullptr ) ingang->info = letter;
}//demo
```

We doen `ingang = nullptr; zetervoor ('A',ingang);`:



Daarna `demo ('B',ingang);` *met* en *zonder* `&` voor de parameter `ingang`. In beide gevallen krijgen we:



Maar nu de volgende functie:

```
void demoanders (vakje*   ingang) {
  ingang = nullptr;
}//demo
```

Met `&` zal de pointer `p` (als deze niet `nullptr` was) bij aanroep `demoanders (p);` veranderen, zonder `&` niet.

NB Dat wat er al dan niet veranderen kan is de *pointer*. Dat waar de pointer naar wijst kan altijd veranderen!

Pointers en arrays hebben veel met elkaar te maken.

Stel dat we hebben `int A[10];` en `int * p;`. Dan kun je het volgende doen:

```
p = A; // p wijst A[0] aan
p++;   // p wijst A[1] aan
p++;   // p wijst A[2] aan
cout << A[2] << " is gelijk aan " << *p << endl;
```

Dus `p` loopt het array langs, en `p++;` gaat naar het volgende array-element, waarbij de grootte van (in dit geval) `int`, `sizeof (int)` dus, gebruikt wordt als "stapgrootte".

Zoeken in pointerlijst

```
class vakje { public:
  char info; vakje* volgende; };//vakje

// Vindt eerste vakje met letter erin (uit lijst met
// ingang), als zo'n vakje bestaat; anders nullptr
vakje* vind (char letter, vakje* ingang) {
  vakje* hulp = ingang;  // NIET eerst hulp = new vakje;
  while ( hulp != nullptr )
    if ( letter == hulp->info )
      return hulp; // of met bool gevonden ...
    else
      hulp = hulp->volgende;
  return nullptr;  // en geen delete's!
}//vind
```

Zoeken in pointerlijst — recursief

```
// Vindt eerste vakje met letter erin (uit lijst met
// ingang), als zo'n vakje bestaat; anders nullptr
// nu recursief
vakje* vindrecursief (char letter, vakje* ingang) {
  if ( ingang == nullptr )
    return nullptr;
  else if ( ingang->info == letter )
    return ingang;
  else // komt letter voor in rest van de lijst?
    return vindrecursief (letter, ingang->volgende);
}//vindrecursief
```

Let op: de functie retourneert een pointer!

Opgave 58

Wat gebeurt er met en zonder &?

```
void tjatja (int* & r, int* & s) {
  r = new int; *r = 1;
  *s = 96;
}//tjatja
int main ( ) {
  int* p; int* q;
  p = new int; *p = 3;
  q = new int; *q = 4;
  cout << *p << *q << endl;
  tjatja (p,q);
  cout << *p << *q << endl;
  return 0;
}//main
```

Met &: 3 4 1 96

Zonder &: 3 4 3(!) 96

In C, dat alleen call by value heeft, moet je `wissel` als volgt schrijven (zie later):

```
void wissel (int *a, int *b) {
  int hulp = *a;  // *a is de int waar a naar wijst
  *a = *b;
  *b = hulp;
}//wissel
```

Voorbeeldaanroep, waarbij `&a` het adres van `a` betekent:
`a = 8; k = 2; wissel (&a,&k);`

NB De functie mag weer `wissel` heten, omdat de types van de parameters anders zijn; dit fenomeen heet overloading.

Merk op dat `a = b; b = a;` niet werkt! Blijkbaar heb je een hulpvariabele nodig.

Of toch niet:

```
void wisseltruc (int & a, int & b) {
  a = a + b;  // a = a_oud + b_oud
  b = a - b;  // b = a_oud
  a = a - b;  // a = b_oud
}//wisseltruc
```

De aanroep `wisseltruc (x,x)` maakt helaas `x` gelijk aan 0. En werkt niet voor bijvoorbeeld strings. En deze getructe functie mag overigens geen `wissel` heten.

Vierde programmeeropgave — 1

Programmeermethoden 2021
Vierde programmeeropgave: Koffiesweeper

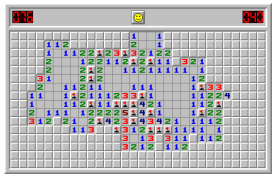
De *vierde* programmeeropgave van het vak **Programmeermethoden** in het najaar van 2021 heet *Koffiesweeper*; zie ook het **elfde werkcollege**, en lees geregeld deze pagina op WWW.

De opgave

Voor deze programmeeropgave gaan we het eenpersoons spel *Koffiesweeper* programmeren, beter bekend als *Minesweeper* (zie **minesweeper.online** en **Wikipedia**). Het is de bedoeling een klasse `koffiebord` te maken, die onder meer memberfuncties heeft als `drukaf`, `menszet` en `randomzet`. Uiteraard heeft deze klasse ook een constructor en een destructor. Verder moeten gedane zetten met behulp van een stapel ongedaan gemaakt kunnen worden.

Maak vanaf het begin gebruik van een aantal voorbeeldfiles, van waaruit de opgave stap voor stap kan worden gedaan. De files zijn allereerst (zie verderop voor gebruik met Code::Blocks, op eigen risico):

- File met main: **hoofd.cc**.
- Headerfile met klassen (en `leesgetal`): **koffiebord.h**.
- Bijbehorende C++-file: **koffiebord.cc**.
- En de bijpassende **makefile** (let op de TABs).



Het spel *Koffiesweeper*, een getrouwe kopie van het welbekende *Minesweeper*, verloopt als volgt. De speler ziet een rechthoek met *m* (hoogte) bij *n* (breedte) vakjes, alle gesloten. Op een aantal vakjes staat (verborgen) een kop koffie of thee. De speler krijgt te horen hoeveel dat er in totaal zijn. De speler kan een vakje selecteren door de coordinaten te geven. Als hier een kop koffie staat heeft de speler onmiddellijk verloren: de koffie moet meteen worden opgedronken. Zo niet, dan ziet de speler een getal tussen 0 en 8: het aantal directe buurvakjes, horizontaal, verticaal en diagonaal, dat een kop koffie bevat.

We spelen het spel als volgt. Eerst mag de grootte van het bord gekozen worden: het aantal rijen *m* en het aantal kolommen *n*, en welk percentage van de vakjes (ongeveer) gevuld is met een kop koffie (gebruik `random ( )` uit `<cstdlib>`; denk aan `srand ( )`; en gebruik de functie `leesgetal` van de derde opgave); het exacte aantal wordt aan de speler bekend gemaakt. De speler kan kiezen of hij/zij zelf één spelletje speelt, of dat er volledig random wordt gespeeld, waarbij het aantal spelletjes gekozen mag worden (steeds met een nieuwe beginconfiguratie). Het eerst geopende vakje bevat nooit een kop koffie.
Als de mens speelt wordt steeds de stand —in eenvoudig formaat— op het scherm getoond, en kan de speler zijn/haar zet doen (een vakje openen), of juist de laatste zet terugnemen (zie straks), of een random zet laten doen, of een vakje markeren met een `'K'`. Als er een reeds eerder geopende plek wordt geselecteerd, moet de speler natuurlijk opnieuw kiezen. Het aantal gedane zetten wordt ook steeds getoond.
De menselijke speler kan een vakje markeren waarvan hij/zij denkt dat het een kop koffie bevat; deze vorm van selecteren telt niet mee voor het aantal zetten. Als een vakje met 0 koffie-buren wordt geopend, kunnen al diens buren veilig worden opgevraagd. Schrijf hiertoe een *recursieve* functie die dit automatisch doet. De speler wint als alle vakjes die geen kop koffie bevatten zijn geopend.

Als het programma volledig random speelt, wordt in een tweetal arrays, één voor de gewonnen en één voor de verloren spelletjes, bijgehouden hoeveel zetten het telkens duurde. Na afloop wordt dan geprint hoeveel spelletjes *z* zetten duurden (*z* = 0, 1, ...), ten behoeve van een grafiek, zie onder.

Schrijf een functie voor de klasse `koffiebord` die een *pointerstructuur* aanlegt, waarbij ieder vakje, naast bijvoorbeeld een `int` en enkele `bool`'s als inhoud, tevens een array met 8 pointers naar de onmiddellijke buren heeft: middenboven (0), rechtsboven (1), rechts (2), rechtsonder (3), middenonder (4), linksonder (5), links (6) en linksboven (7). De vakjes aan de randen bevatten uiteraard diverse `nullptr`'s. Het bord is dus *niet* een *m* bij *n* array, maar een zeer ingewikkelde pointerstructuur.

Bij de menselijke speler moeten alle complete borden op een *stapel* worden bijgehouden, en deze kunnen daarmee teruggenomen worden. Zodra een speler zet, wordt een kopie van het bord opgeslagen. Dit onderdeel is zeker niet eenvoudig; mocht het ontbreken, dan kost dat een punt.

Het is de bedoeling om een vijftal files te produceren: de eerste bevat `main` en het menutje, de tweede (zeg `koffiebord.h`, zie boven) bevat de klasse-definitie voor `koffiebord`, en de derde (zeg `koffiebord.cc`, zie boven) bevat de functies uit die klasse. Evenzo zijn er files `stapel.h` en `stapel.cc`, indien van toepassing. Maak ook een `makefile`.
Code::Blocks-gebruikers: doe deze opgave liever op een Linux-machine. Maar het kan wel: open een nieuw project via "File -- New project -- Empty project", vul wat in, en voeg de drie files toe via "Project -- Add files", en daarna het project compileren op de gebruikelijke manier. Of, op eigen risico, lees **over projecten**.

Opmerkingen
Gebruik geschikte (member)functies. Bij deze opgave mogen wederom bij elke functie tussen begin-{ en eind-} *hooguit circa 30* niet al te volle of complexe regels staan! Elke functie dient van commentaar voorzien te zijn. Als uitzondering mag `main` langer zijn, als daarin met het menu gewerkt wordt. Let op goed parametergebruik: alle parameters, met uitzondering van membervariabelen, in de heading doorgeven, en de variabele-declaraties zowel bij `main` als bij de andere functies aan het begin. De enige te gebruiken headerfile is in principe `iostream`, en eventueel `ctime` en `cstdlib` (voor de random-generator). Zeer ruwe indicatie voor de lengte van de gezamenlijke C++-files: 600 regels. Denk aan het infoblokje.

Uiterste inleverdatum: **maandag 13 december 2021, 17:00 uur**.
Manier van inleveren:

1. Digitaal de C++-code **inleveren**: stuur een email naar `pm@liacs.leidenuniv.nl`.
   Stuur geen executable's, lever alleen de vijf (of drie) C++-files en de `makefile` digitaal in! Noem de file `koffiebord.cc` hier bij voorkeur zoiets als `garfunkelsimongobord4.cc`, dit voor de opdracht van het duo Simon-Garfunkel, en analoog de andere files. De laatst voor de deadline ingeleverde versie wordt nagekeken.
2. En *ook* een papieren versie van het verslag (inclusief de C++-code van alle files, en de `makefile`) deponeren in de speciaal daarvoor bestemde doos "Programmeermethoden" bij kamer 159 van het Snellius-gebouw.
   Overal duidelijk datum en namen van de (maximaal twee) makers vermelden, in het bijzonder als commentaar in de eerste regels van de C++-code.
   Het *verslag* (uiteraard weer in LaTeX, zie de eerdere opgaven) moet het volgende bevatten: een zeer korte beschrijving van het programma, een beschrijving van punten waarop het programma faalt (indien van toepassing), en een tabel met gewerkte uren, uitgesplitst per week en per persoon. En een referentie betreffende Minesweeper. En een grafiek (zie het **bijbehorende werkcollege** voor tips) waarin staat hoe lang een winnend of verliezend spelletje voor de random speler duurt — voor verschillende beginconfiguraties.

Te gebruiken compiler: als hij maar C++ vertaalt; het programma moet in principe zowel op een Linux-machine (met `g++`) als onder Visual C++ of Code::Blocks draaien. Test dus in principe op beide systemen! Het programma wordt doorgaans nagekeken met behulp van de compiler die (uiteraard) in het commentaar bovenin het programma vermeld staat. Normering: layout 1; grafiek 1; verslag 1; commentaar 1; modulariteit (OOP, functies) 2; werking 4. Eventuele aanvullingen en verbeteringen: lees de huidige WWW-bladzijde: `www.liacs.leidenuniv.nl/~kosterswa/pm/op4pm.php`.

www.liacs.leidenuniv.nl/~kosterswa/pm/op4pm.php

Koffiesweeper programmeren we als volgt:

- week 1 (“10”): pointerpracticum, opgave lezen
- week 2 (“11”): klassen, pointerstructuur aanleggen, elementair spelen
- week 3 (“12”): finesses, recursie, stapel
- week 4 (“13”): experiment (`gnuplot`), verslag

`www.liacs.leidenuniv.nl/~kosterswa/pm/op4pm.php`

- lees de vierde programmeeropgave, denk na over de klassen; de deadline is op maandag 13 december 2021
- lees Savitch Hoofdstuk 10
- lees dictaat Hoofdstuk 3.12
- maak opgaven 44/46, 52/56 uit het opgavendictaat
- doe het pointerpracticum: werkcollege 10
- `www.liacs.leidenuniv.nl/~kosterswa/pm/`

Programmeermethoden

Recursie

Walter Kosters en Jonathan Vis

week 11: 22–26 november 2021

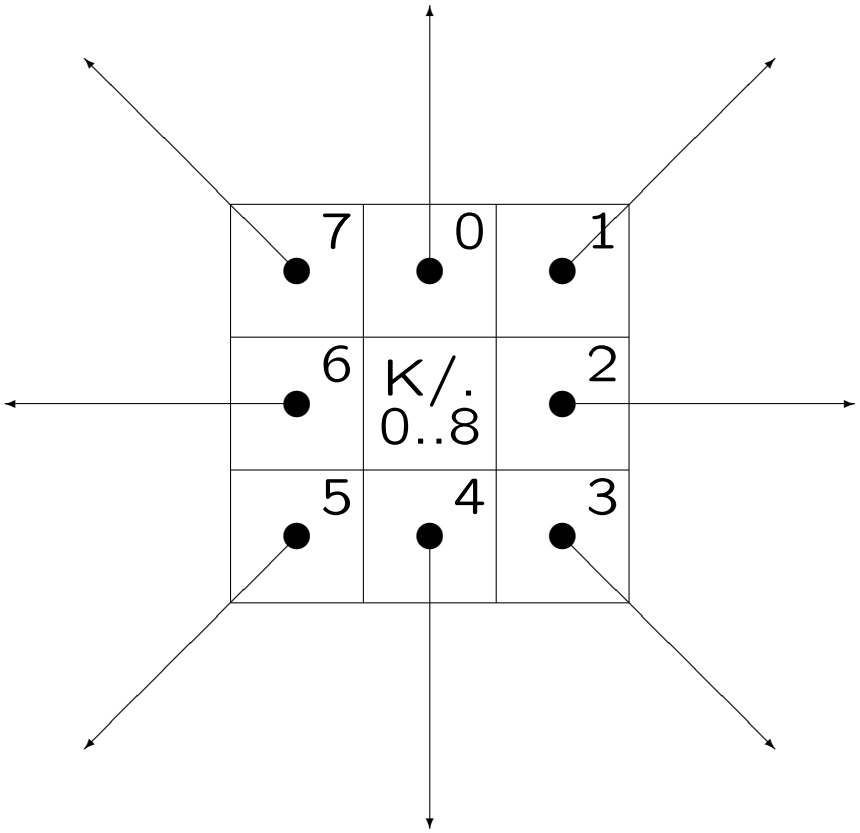
www.liacs.leidenuniv.nl/~kosterswa/pm/

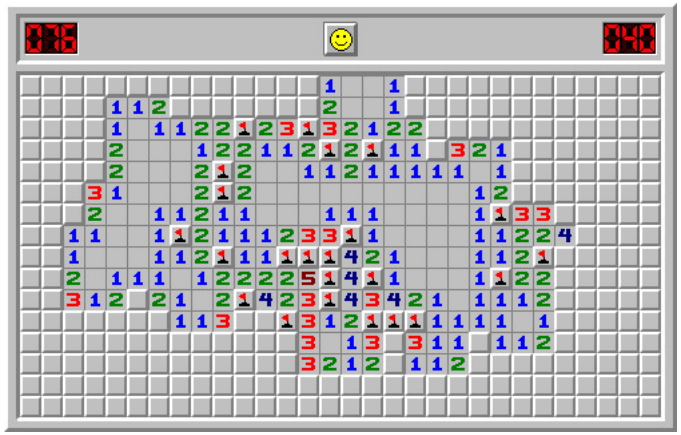
Koffiesweeper programmeren we als volgt:

- week 1: pointerpracticum, opgave lezen
- week 2: klassen, pointerstructuur aanleggen, elementair spelen
- week 3: finesses, recursie, stapel
- week 4: experiment (`gnuplot`), verslag

www.liacs.leidenuniv.nl/~kosterswa/pm/op4pm.php

Vierde programmeeropgave — 2





Vierde programmeeropgave — 3

Als je C++-code over meerdere files verdeelt, helpt een `makefile` bij het compileren (aanroep: `make`). Stel je hebt:

file `ks.h`

```
class ks {
  ...
  void print ( );
};//ks
```

file `ks.cc`

```
#include <iostream>
#include "ks.h"
// implementatie van
// prototypes uit
// ks.h
void ks::print ( ) {
  ...
}//ks::print
```

file `hoofd.cc`

```
#include <iostream>
#include "ks.h"
...
int main ( ) {
  ks X;
  X.print ( );
  ...
}//main
```

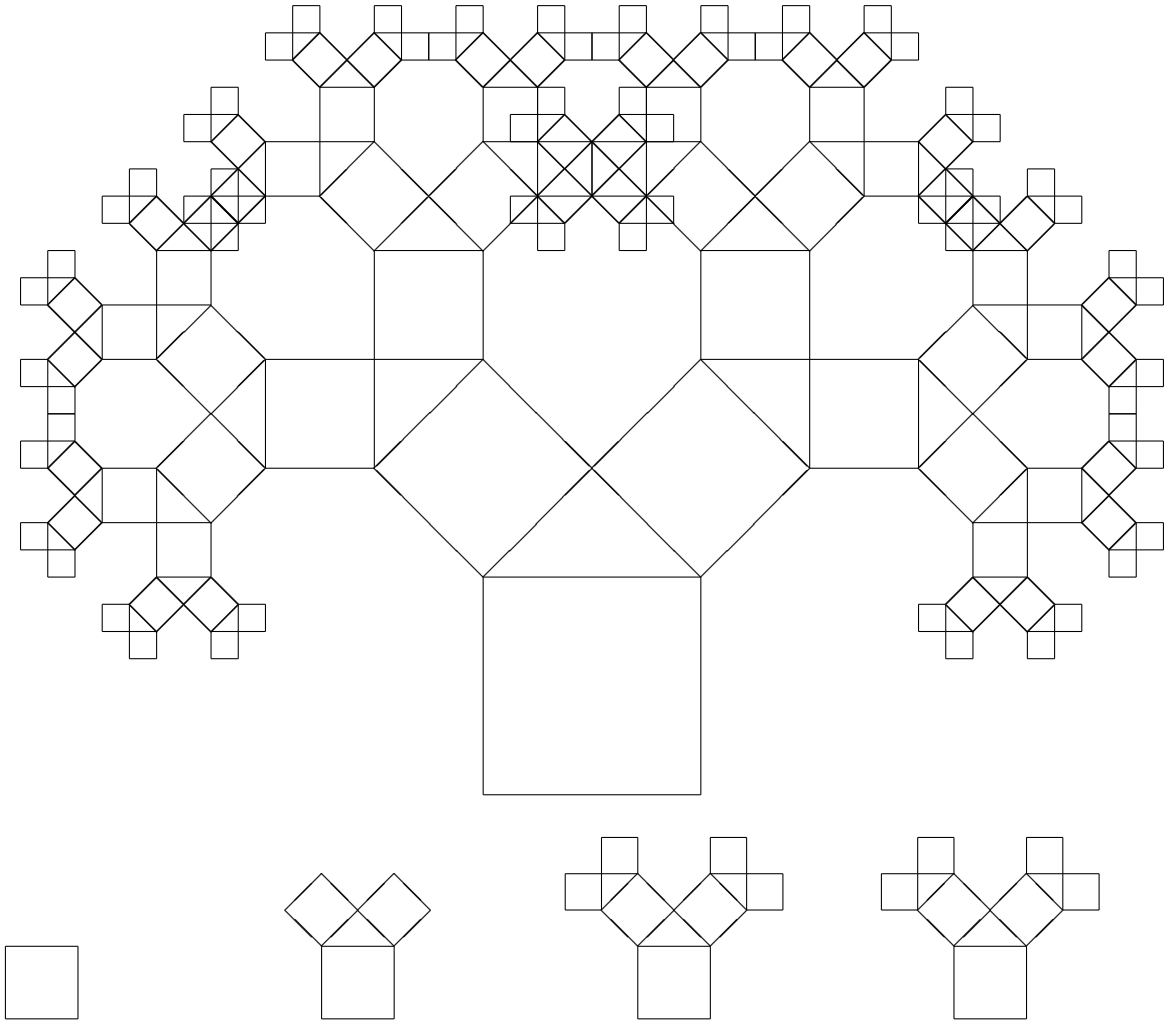
De makefile ziet er dan bijvoorbeeld uit als (let op tabs!):

```
all: ks.o hoofd.o
←TAB→g++ -Wall -Wextra -o koffie ks.o hoofd.o
ks.o: ks.cc ks.h
←TAB→g++ -Wall -Wextra -c ks.cc
hoofd.o: hoofd.cc ks.h
←TAB→g++ -Wall -Wextra -c hoofd.cc
```

in Code::Blocks via projecten, zie video

Vierde programmeeropgave — 4

```
// file ks.h
class bordvakje {
  public:  bordvakje* buren[8];
           int info; // ... TODO
};//bordvakje

class ks {
  private: bordvakje* ingang;
  public:  void print ( );
           // ... TODO
};//ks

// file ks.cc
void ks::print ( ) { ... }
```



Boom van Pythagoras

Basisidee:

proces is recursief als het naar zichzelf verwijst

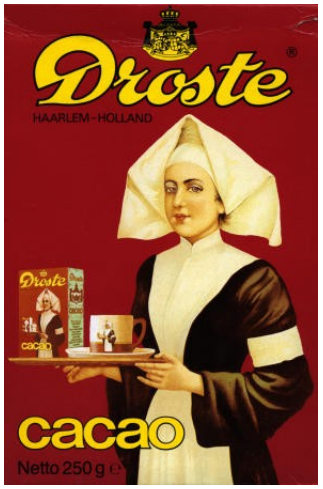
functie is recursief als deze zichzelf (in)direct aanroept

Voorbeeld:

$$S(n) = \sum_{i=1}^{n} i^{42} = \sum_{i=1}^{n-1} i^{42} + n^{42} = S(n-1) + n^{42}$$

Woordenboek:

Recursie: zie Recursie
Fiets: zie Rijwiel
Rijwiel: zie Fiets



Een recursief/ve proces/procedure/functie bestaat in het algemeen uit twee delen:

1. één of meer klein(st)e (eenvoudig(st)e) gevallen die direct oplosbaar zijn: de basisgevallen

2. een algemene methode die een bepaald geval reduceert tot één of meer kleinere (eenvoudiger) gevallen, waarbij men uiteindelijk op een basisgeval uitkomt

Algemene gedaante van een recursieve functie:

if basisgeval **then**

los op zonder recursie; // makkelijk

else

één of meer recursieve eenvoudigere aanroepen;

fi

Let op de symbolische notatie in pseudo-code.

Probleem:

Betaal(X) = Betaal het bedrag X

Oplossing:

Betaal(0) = Doe niets

Betaal(X) = Geef de grootste munt $Y \leqslant X$

Betaal daarna $X - Y$: Betaal($X - Y$)

Vraag:

Wat kan er nog fout gaan? betaal 30 met 25/10 (vóór 2002)

Recursieve definitie van n-faculteit ($n!$):

$$\text{fac}(n) = \begin{cases} 1 & \text{als } n = 0 \\ n \times \text{fac}(n-1) & \text{als } n > 0 \end{cases}$$

Recursieve C++-functie ($\texttt{n} \geqslant 0$):

```
long faculteit (int n) {
  if ( n == 0 ) // basisgeval
    return 1;
  else
    return n * faculteit (n-1); // recursie
}//faculteit
```

Een functie mag zichzelf (in)direct aanroepen: recursie.

```
int som (int n) { // berekent 1 + 2 + ... + n   versie 1
  int i, res = 0;
  for ( i = 1; i <= n; i++ ) res += i;
  return res;
}//som

int somrecursief (int n) { // idem, recursief   versie 2
  if ( n == 0 ) return 0;
  else return n + somrecursief (n-1);
}//somrecursief

int somslimGauss (int n) { // en nog eens ...   versie 3
  return ( n * ( n + 1 ) ) / 2;
}//somslim
```

De ggd kan ook recursief berekend worden:

```
int ggdrecursief (int x, int y) {
  if ( y == 0 ) return x;
  else return ggdrecursief (y,x % y);
}//ggdrecursief
```

Je gebruikt eigenlijk:

$$\text{ggd}(x,y) = \begin{cases} x & \text{als } y = 0 \\ \text{ggd}(y, x \bmod y) & \text{als } y \neq 0 \end{cases}$$

Definitie Fibonacci-getallen:

$$\text{fib}(n) = \begin{cases} 1 & \text{als } n = 0 \text{ of } n = 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{als } n > 1 \end{cases}$$

Alternatief: $\text{fib}(1) = \text{fib}(2) = 1$.

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, ...

Fibonacci — recursief

Recursieve C++-functie:

```
long fib1 (int n) {
  if ( ( n == 0 ) || ( n == 1 ) )
    return 1;
  else
    return ( fib1 (n-1) + fib1 (n-2) );
}//fib1
```

Er is hier sprake van een watervaleffect:
de aanroep `fib1 (5)` veroorzaakt
14 andere (dubbele) aanroepen.

Fibonacci — Dynamisch programmeren

```
const int MAX = 100;
long memo[MAX]; // globaal, initialiseer met 0-en!

// recursie met array
long fib2 (int n) {
  if ( n >= MAX ) // helaas
    return fib2 (n-1) + fib2 (n-2);
  else
    if ( memo[n] > 0 ) // al eerder berekend
      return memo[n];
    else {
      if ( ( n == 0 ) || ( n == 1) )
        memo[n] = 1;
      else
        memo[n] = fib2 (n-1) + fib2 (n-2);
      return memo[n];
    }//else
}//fib2
```

Iteratief: opsommen tot je bij fib(n) bent:

```
long fib3 (int n) {
  long eerste = 1, tweede = 1, hulp;
  int teller;
  for ( teller = 2; teller <= n; teller++ ) {
  // nu geldt: eerste == fib (teller-2) en
  // tweede == fib (teller-1)  ("invariant")
     hulp = tweede;
     tweede = eerste + tweede;
     eerste = hulp;
  }//for
  return tweede;
}//fib3
```

Deze versie is erg geschikt voor “grote getallen”.

Gesloten formule (nauwelijks te berekenen!):

$$\texttt{fib}(n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right)$$

↑

klein in absolute waarde

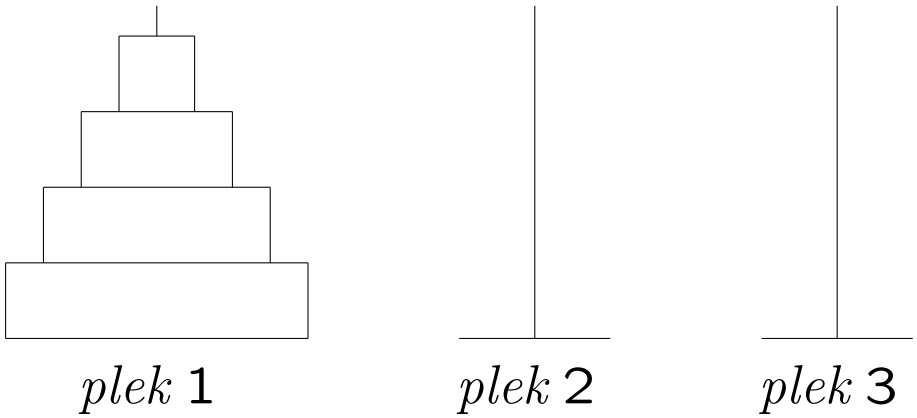
Met matrices:

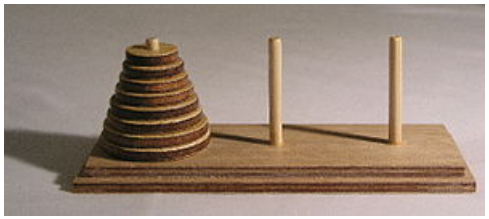
$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} \texttt{fib}(n) & \texttt{fib}(n-1) \\ \texttt{fib}(n-1) & \texttt{fib}(n-2) \end{pmatrix}$$

Torens van Hanoi — 1

Gegeven: n ($n \geqslant 1$) schijven met gat in het midden, alle verschillend in grootte, en 3 palen = plekken
Beginsituatie: alle schijven liggen boven op elkaar om één paal, en de andere 2 palen zijn leeg
Restrictie: een grotere schijf ligt nooit op een kleinere

Voorbeeld: beginsituatie voor $n = 4$

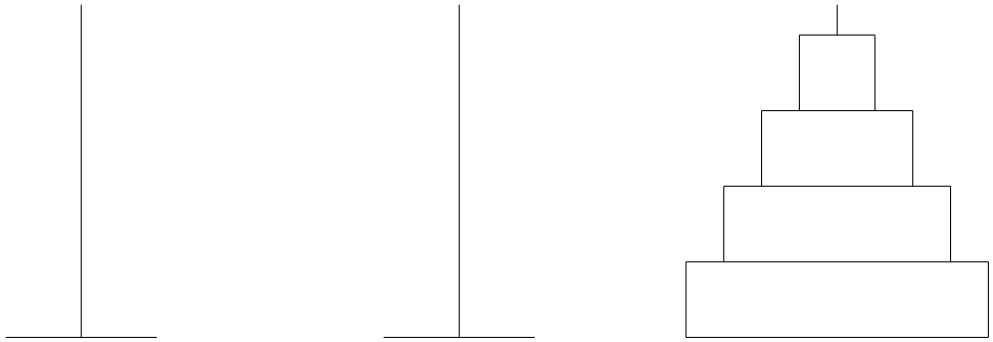




Torens van Hanoi — 2

Doel: breng de hele toren naar een van de lege palen
Acties: per keer mag je één schijf verplaatsen (de bovenste van een stapel), en deze bovenop een andere stapel leggen
Restrictie: er mogen alleen kleinere schijven op grotere worden gelegd

Voorbeeld: eindsituatie voor $n = 4$



Oplossing:

n schijven zo efficiënt mogelijk van start naar doel verplaatsen (via hulp) =

> eerst de bovenste $n-1$ *schijven zo efficiënt mogelijk van start naar hulp verplaatsen (via doel)*,
>
> dan de grote schijf van start naar doel,
>
> en tenslotte de $n-1$ *schijven zo efficiënt mogelijk van hulp naar doel verplaatsen (via start)*

Dat is recursie! Wat is het basisgeval?

Algoritme:

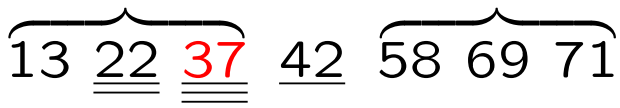
```
// Torens van Hanoi: recursief
// zet toren van n stuks (optimaal) van a naar b via c
// print de zetten
void zet (int n, int a, int b, int c) {
  if ( n > 0 ) {
    zet (n-1,a,c,b);
    cout << "Zet van " << a << " naar " << b << endl;
    zet (n-1,c,b,a);
  }//if
}//zet
```

Het aantal zetten is in totaal $2^n - 1$.

Aanroep: `zet (aantal,1,3,2);`, waarbij `aantal` het gewenste aantal schijven is. Basisgeval: "niets doen".

Probleem: Zoek een waarde in een *oplopend gesorteerd* array A met n elementen

Oplossing: binair zoeken



- Kijk of het middelste (†) element het gezochte is.
- Stop indien het element gevonden is, of als het te onderzoeken array leeg is.
- Anders: bepaal op grond van vergelijken met dat middelste element of verder (recursie!) gezocht moet worden in de linker helft óf in de rechter helft van het array en herhaal dit.

(†) Als het aantal elementen even is: kies één van de twee middelste.

Binair zoeken: iteratief

```
// Geeft index met A[index] = getal, als getal voorkomt;
// zo niet: resultaat wordt -1.

int binairzoeken (int A[ ], int n, int getal) {
  int links = 0, rechts = n-1;  // zoek tussen links en rechts
  int midden;

  while ( links <= rechts ) {
    midden = ( links + rechts ) / 2;
    // beter(!): midden = links + ( rechts - links ) / 2;
    if ( getal == A[midden] )
      return midden;  // of gevonden = true etc.
    else if ( getal > A[midden] )
      links = midden + 1;
    else
      rechts = midden - 1;
  }//while

  return -1;
}//binairzoeken
```

Binair zoeken: recursief

```
int binairzoeken (int A[ ], int n, int links, int rechts, int getal) {
  int midden;
  if ( links > rechts )           // basisgeval: leeg interval
    return -1;                    // dus stop; niet aanwezig
  else {                          // nu echt zoeken
    midden = ( links + rechts ) / 2;
    if ( getal == A[midden] )     // gevonden!
      return midden;
    else                  // verder zoeken: recursieve aanroepen
      if ( getal > A[midden] )    // rechts hetzelfde doen
        return binairzoeken (A,n,midden+1,rechts,getal);
      else                        // links hetzelfde doen
        return binairzoeken (A,n,links,midden-1,getal);
  }//else echt zoeken
}//binairzoeken
```

Aanroep: `iets = binairzoeken (A,n,0,n-1,getal);`

Recursief sorteren

```
sorteer (rij) =
  if ( rij heeft meer dan 1 element ) then
    verdeel rij in linkerrij en rechterrij;
    sorteer (linkerrij);
    sorteer (rechterrij);
    combineer (linkerrij, rechterrij);
  fi
```

⇓ (zie elders)

Mergesort: $O(n \lg n)$

Quicksort: $O(n \lg n)$

Insertion sort: $O(n^2)$

$n =$ aantal elementen van de rij; $\lg n = {}^2\log n = \log_2 n$

Recursie en parameters 1 ⋈

↑
geen tentamenstof

```
void print1 (int    a) {  // call by value
  if ( a > 0 ) {
    a--;
    print1 (a);
    cout << a << ", ";

  }//if
}//print1
```

Nu doen we:

```
getal = 3; print1 (getal); cout << getal << endl;
```

Dat levert: `0, 1, 2, 3`

Recursie en parameters 2 ⋈

```
void print2 (int & a) {  // call by reference
  if ( a > 0 ) {
    a--;
    print2 (a);
    cout << a << ", ";

  }//if
}//print2
```

Nu doen we:

```
   getal = 3; print2 (getal); cout << getal << endl;
```

Dat levert: `0, 0, 0, 0`

```
void print3 (int & a) {  // call by reference
  if ( a > 0 ) {
    a--;
    print3 (a);
    cout << a << ", ";
    a++; // en a weer terugzetten
  }//if
}//print3
```

Nu doen we:

```
   getal = 3; print3 (getal); cout << getal << endl;
```

Dat levert: `0, 1, 2, 3`

Recursie wordt ook vaak gebruikt bij het programmeren van spellen als Schaken, Go en Boter, kaas en eieren.

We willen het aantal vervolgpartijen $S.Aantal\ (\ )$ weten vanuit een gegeven stand (= positie) S:

```
S.Aantal ( ) ::
    Teller ← 0;
    if S is eindstand then
        return 1;
    fi
    for alle mogelijke zetten z do
        S.DoeZet (z);
        Teller ← Teller + S.Aantal ( );
        S.OntDoeZet (z);
    od
    return Teller;
```

Bij deze oplossing is ervoor gekozen de Stand S niet "kapot" te maken, vandaar de aanroep $OntDoeZet\ (z)$. Gebruik makend van de eigenschap dat recursieve aanroepen S niet verstoren, doet de buitenste aanroep dat nu ook niet.

Je kunt ook, voor iedere z opnieuw, de zet doen in een kopie van S, zodat je S nooit vernielt.

Overigens: er zijn 255.168 verschillende *partijen* Boter, kaas en eieren. En je hebt dan meteen het hele spel doorgerekend (zie later).

Opgave 1 van het tentamen van 6 januari 2020:

In het array `int A[n]` staan `n` (een `const int` $\geqslant 3$) verschillende gehele getallen.

a. (6) Schrijf een Booleaanse C++-functie `gem (A,n)` die `true` geeft als er precies één array-element in `A` is (niet eerste of laatste) dat *exact* het gemiddelde is van zijn beide directe buren, en anders `false`. Dus `true` voor array `10 8 6 1`, en `false` voor `2 5 9`.

b. (7) Schrijf een C++-functie `int stijg (A,b,n)` die de lengte van een langste stijgende aaneengesloten deelrij van `A` geeft, en diens begin-index in `b` oplevert. Als er meerdere deelrijen het langste zijn: de kleinste `b`. Dus array `2 7 4 5 6 1 3 8` geeft 3, met `b` = 2 (deelrij `4 5 6`, even lang als `1 3 8`).

c. (4) We nemen in dit onderdeel aan dat `A` uit precies twee stijgende aaneengesloten deelrijen bestaat. Schrijf een C++-functie `int kl (A,n)` die het kleinste element van `A` oplevert, door de functie van **b** te gebruiken, en dan de twee kandidaten te vergelijken.

d. (4) Schrijf een C++-functie `bu (A,n)` die `A` *aflopend* sorteert met *bubblesort*. De functie moet stoppen als er tijdens een doorgang/ronde geen verwisselingen waren.

e. (4) Hoeveel vergelijkingen tussen array-elementen doet de functie van **d** minimaal en maximaal, uitgedrukt in `n`? En wanneer gebeurt dat?

Uitwerking 6.1.2020: Opgave 1

```
a. bool gem (int A[ ], int n) {
     int i, tel = 0;
     for ( i = 1; i < n-1; i++ )
       if ( A[i-1] + A[i+1] == 2 * A[i] ) tel++;
     return ( tel == 1 );
   }//gem
b. int stijg (int A[ ], int & b, int n ) {
     int i, langste = 1, lang = 1, begin = 0; b = 0;
     for ( i = 1; i < n; i++ )
       if ( A[i] > A[i-1] ) {
         lang++; if ( lang > langste ) { langste = lang; b = begin; }
       }//if
       else { begin = i; lang = 1; }//else
     return langste;
   }//stijg
c. int kl (int A[ ],int n) {
     int b, s; s = stijg (A,b,n); if ( b == 0 ) b = s;
     if ( A[0] < A[b] ) return A[0]; else return A[b];
   }//kl
d. void bu (int A[ ], int n) {
     int i, j = 0, tmp; bool wissel = true;
     while ( wissel ) {
       wissel = false; j++;
       for ( i = 0; i < n-j; i++ ) // of i < n-1 (*)
         if ( A[i] < A[i+1] ) { // aflopend sorteren
   tmp = A[i]; A[i] = A[i+1]; A[i+1] = tmp; wissel = true; }}//if&while
   }//bu
e. Al aflopend gesorteerd: n-1 vergelijkingen; omgekeerd gesorteerd:
   n-1 + n-2 + ... + 1 = n(n-1)/2 = O(n^2)  (bij (*): n(n-1))
```

- maak de vierde programmeeropgave — de deadline is op maandag 13 december 2021
- lees Savitch Hoofdstuk 13
- lees dictaat Hoofdstuk 3.10, 4.2.2, 4.2.7
- maak opgaven 57/61 uit het opgavendictaat
- www.liacs.leidenuniv.nl/~kosterswa/pm/

Programmeermethoden

Datastructuren: stapels, rijen en binaire bomen

Walter Kosters en Jonathan Vis

week 12: 29 november–3 december 2021

www.liacs.leidenuniv.nl/~kosterswa/pm/

Koffiesweeper programmeren we als volgt:

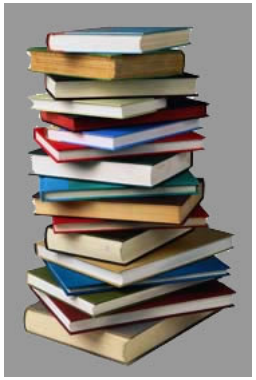
- week 1: pointerpracticum, opgave lezen
- week 2: klassen, pointerbord aanleggen, elementair spelen
- week 3: finesses, recursie, stapel
- week 4: experiment (`gnuplot`), verslag

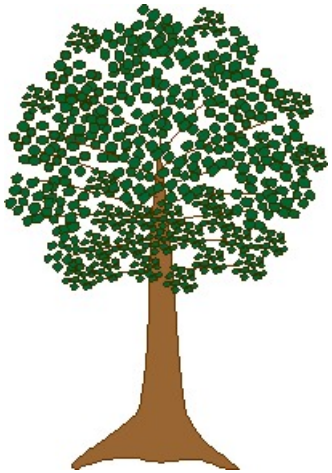
www.liacs.leidenuniv.nl/~kosterswa/pm/op4pm.php

In de informatica worden Abstracte DataTypen (ADT's) zoals stapels, rijen en bomen veelvuldig gebruikt.

Bij de afwikkeling van recursie komen bijvoorbeeld stapels goed van pas.

De OOP-filosofie sluit hier mooi op aan.





Een DataType bestaat uit een domein (een collectie "waarden", al dan niet met structuur), in combinatie met een aantal basisoperaties die op dit domein gedefinieerd zijn.

We spreken van een Abstract DataType als de implementatie van de operaties is afgeschermd van de gebruiker.

Voorbeeld 1: De gehele getallen (`int` in C++), met basisoperaties zoals $+$, $-$ en $*$.
De gebruiker kan deze operaties wel gebruiken, maar weet niet (en hoeft ook niet te weten) hoe deze precies in C++ zijn geïmplementeerd.

Voorbeeld 2: Verzamelingen, zoals verzamelingen gehele getallen tussen 0 en n: $\{3, 6, 10\}$.

Bekijk het datatype Verzameling (Set) waarvan het domein bestaat uit verzamelingen gehele getallen tussen 0 en n. Een verzameling is ongeordend en bevat allemaal verschillende elementen. Als basisoperaties op deze verzamelingen definiëren we:

- een lege verzameling aanmaken (of een bestaande leeg maken),
- kijken of de verzameling leeg is,
- testen of een gegeven getal erin zit,
- een getal eraan toevoegen,
- een getal eruit halen.

Klasse verzameling — 1

```
class verzameling {
  public:
    verzameling ( );          // constructor
    bool isleeg ( );          // is V leeg?
    bool ziterin (int i);     // zit i in V ?
    void erbij (int i);       // stop i in V
    void eruit (int i);       // haal i uit V
    ...
  private:       // de implementatie wordt afgeschermd
    bool inhoud[n];           // n een constante
};//verzameling
```

We implementeren een verzameling V dus met behulp van een array `inhoud`, waarbij `inhoud[i] == true` $\Longleftrightarrow i \in V$.

Met behulp van de basisoperaties kunnen we andere functies schrijven, zoals `void verzameling::doorsnede (A,B)`.

Klasse verzameling — 2

```
verzameling::verzameling ( ) {
  for ( int i = 0; i < n; i++ ) inhoud[i] = false;
}//verzameling::verzameling
bool verzameling::isleeg ( ) {
  for ( int i = 0; i < n; i++ )
    if ( inhoud[i] ) return false;
  return true;
}//verzameling::isleeg
bool verzameling::ziterin (int i) {
  return inhoud[i];
}//verzameling::ziterin
void verzameling::erbij (int i) {
  inhoud[i] = true;
}//verzameling::erbij
void verzameling::eruit (int i) {
  inhoud[i] = false;
}//verzameling::eruit
```

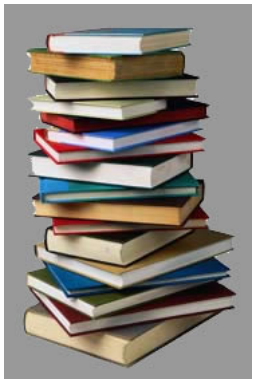
Klasse verzameling — 3

```
// de verzameling *this wordt gelijkgemaakt aan A door B
void verzameling::doorsnede (verzameling & A,
                             verzameling & B) {
  for ( int i = 0; i < n; i++ ) {
    if ( A.ziterin (i) && B.ziterin (i) )
      erbij (i);  // oftewel this->erbij (i);
  }//for
}//doorsnede
```

In main ("emuleert" {6,10} $\cap$ {3,6} = {6}):

```
  verzameling een, twee, drie;
  een.erbij (10); een.erbij (6);  // vul een = {6,10}
  twee.erbij (3); twee.erbij (6); // vul twee = {3,6}
  drie.doorsnede (een, twee);     // drie wordt de doorsnede
                                  // van een en twee: {6}
```

Een stapel (stack; denk aan een stapel boeken) is een reeks elementen van hetzelfde type, bijvoorbeeld gehele getallen, met de volgende toegestane operaties:

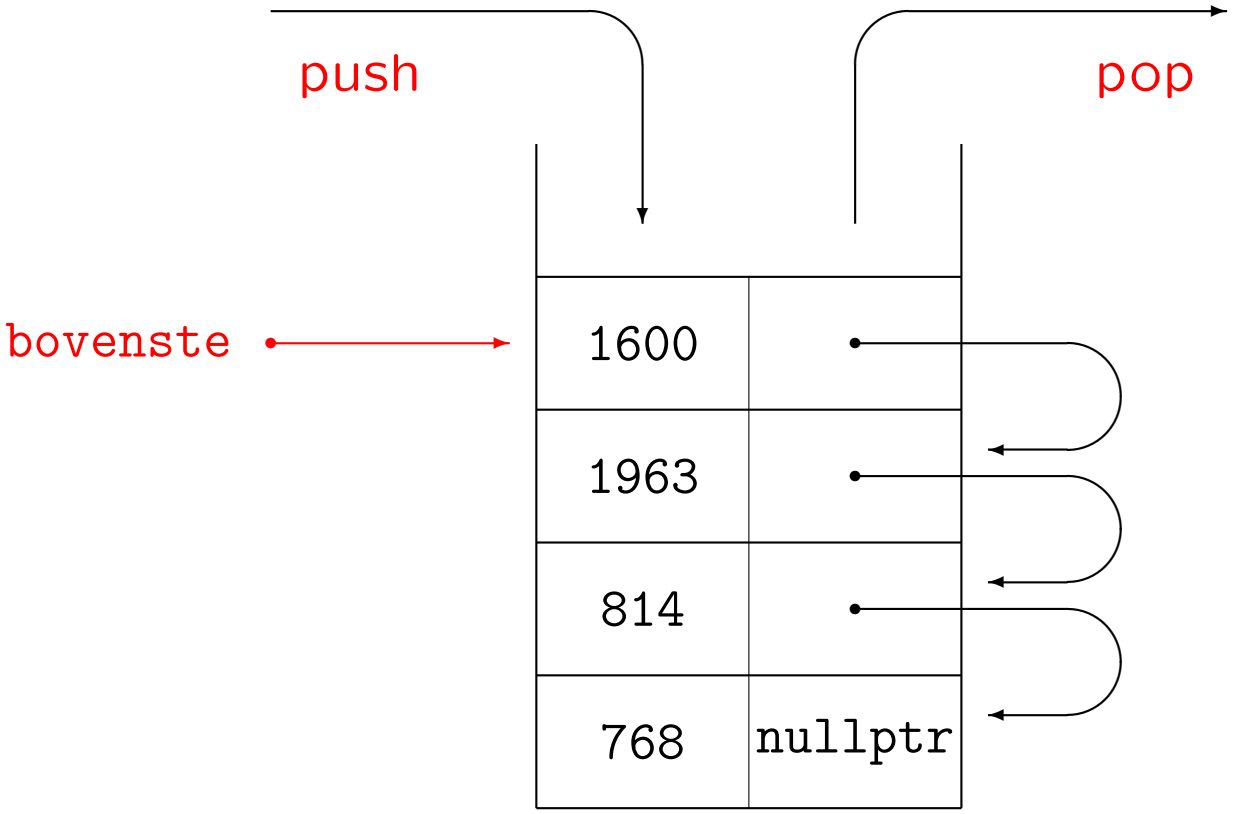


- een lege stapel aanmaken,
- testen of de stapel leeg is,
- een element toevoegen (*push*),
- het laatst-toegevoegde element eruithalen (*pop*),
- soms: kijken of de stapel al vol is.

Een stapel heeft dus de *LIFO-eigenschap*: LIFO = Last In First Out. Toevoegen en verwijderen gebeurt derhalve aan dezelfde kant: de *bovenkant*.

```
class stapel {  // stapel die gehele getallen bevat
  public:
    stapel ( );
    bool isstapelleeg ( );
    void zetopstapel (int);        // push
    void haalvanstapel (int&);     // pop, let op &
    ...
  private:
    // implementatie met pointers of array
};//stapel
```

C++ aanroep:	abstracte notatie:
`S.zetopstapel (768);`	S ⇐ 768;
`S.haalvanstapel (jaartal);`	jaartal ⇐ S;


push
pop
bovenste
1600
1963
814
768
nullptr

We hebben een extra type nodig voor de vakjes waaruit de pointerlijst bestaat. De vakjes zijn opgebouwd uit een veld `info` voor een geheel getal en een veld `volgende` voor de rest van de stapel.

```
class vakje {  // een struct mag ook
  public:
    // constructor (een destructor hoeft misschien niet)
    vakje ( ) {                          // beter niet "inline"
      info = 0; volgende = nullptr; }// constructor vakje
    int info;
    vakje* volgende;
};//vakje
```

De stapel als enkelverbonden lijst:

```
class stapel {   // de stapel zelf
  public:
    stapel ( ) {
      bovenste = nullptr; }// maak lege stapel
    ~stapel ( );   // destructor
    void zetopstapel (int);       // push
    void haalvanstapel (int&);    // pop
    bool isstapelleeg ( ) {       // is stapel leeg?
      return ( ( bovenste == nullptr ) ? true : false );
      // of: if ( bovenste == nullptr ) ...
    }//isstapelleeg
    ...
  private:                 // het begin van de lijst is
    vakje* bovenste;       // de bovenkant van de stapel
};//stapel
```

```
void stapel::zetopstapel (int getal) {        // push
  vakje* temp = new vakje;
  temp->info = getal;
  temp->volgende = bovenste;
  bovenste = temp;
}//stapel::zetopstapel

void stapel::haalvanstapel (int & getal) {    // pop
  vakje* temp = bovenste;
  getal = bovenste->info;
  bovenste = bovenste->volgende;
  delete temp;
}//stapel::haalvanstapel
```

NB Bij deze `haalvanstapel` hoef je er niet op te letten of de stapel leeg is, dat moet de gebruiker via `isstapelleeg` zelf maar doen . . .

En de destructor die de pointerlijst netjes afbreekt:

```
stapel::~stapel ( ) {
  int getal;
  while ( ! isstapelleeg ( ) )
    haalvanstapel (getal);
}//stapel::~stapel
```

Deze destructor wordt "vanzelf" aangeroepen als de betreffende variabele ophoudt te bestaan, dus aan het eind van de functie waarin de variabele gedeclareerd is.

Vaak wordt hiervan een aparte `verwijder`-functie gemaakt, met destructor: `stapel::~stapel ( ) { verwijder ( ); }`

Stapel via een array — 1

```
const int MAX = 100;
class stapel { // voor maximaal MAX integers
  public:
    stapel ( ) { bovenste = -1; }  // constructor
    void zetopstapel (int);
    void haalvanstapel (int&);
    bool isstapelleeg ( ) {
      return ( bovenste == -1 ); }
    ...
  private:
    int inhoud[MAX];
    int bovenste;     // index bovenste waarde
};//stapel
```

Stapel via een array — 2

```
void stapel::zetopstapel (int getal) {
  bovenste++;
  inhoud[bovenste] = getal;
}//stapel::zetopstapel

void stapel::haalvanstapel (int & getal) {
  getal = inhoud[bovenste];
  bovenste--;
}//stapel::haalvanstapel
```

Er is eigenlijk ook een memberfunctie `vol` nodig, bijvoorbeeld in het `private`-gedeelte gedefinieerd. Deze functie wordt dan in `zetopstapel` aangeroepen.

```
bool stapel::vol ( ) {
  return ( bovenste == MAX - 1 );
}//stapel::vol
```

```
void haalgrootstegetaluitstapel (stapel & S, int & grootste) {
  stapel hulp;
  int x;
  if ( ! S.isstapelleeg ( ) ) {
    S.haalvanstapel (grootste);
    hulp.zetopstapel (grootste);
    while ( ! S.isstapelleeg ( ) ) {
      S.haalvanstapel (x);
      if ( x > grootste )
        grootste = x;
      hulp.zetopstapel (x);
    }//while
    while ( ! hulp.isstapelleeg ( ) ) {
      hulp.haalvanstapel (x);
      if ( x != grootste )
        S.zetopstapel (x);
    }//while
  }//if
}//haalgrootstegetaluitstapel
```

Merk op dat de precieze implementatie van de stapel er niet toe doet, evenmin als in het volgende voorbeeld.

Stapel: gebruik in main

```
int main ( ) {   // een main die de stapel gebruikt
  stapel S;
  int getal = 0;
  while ( getal >= 0 ) {   // zet getallen > 0 op stapel
    S.drukaf ( );   // nog te schrijven memberfunctie
    cout << "getal > 0: push; = 0: pop; < 0 stop" << endl;
    cin >> getal;
    if ( getal > 0 )
      S.zetopstapel (getal);
    else
      if ( ( getal == 0 ) && ( ! S.isstapelleeg ( ) ) ) {
        S.haalvanstapel (getal);
        cout << getal << " van stapel gehaald " << endl;
      }//if
  }//while
  return 0;
}//main
```

In de Standard Template Library (STL) zitten ook al complete stapels ("stacks"):

```
#include <stack>

stack<int> S;
S.push (1994); S.push (2021);
while ( ! S.empty ( ) ) {
  cout << S.top ( ) << endl; S.pop ( ); }//while
```

Tussen < > staat het soort elementen dat op de stapel komt.

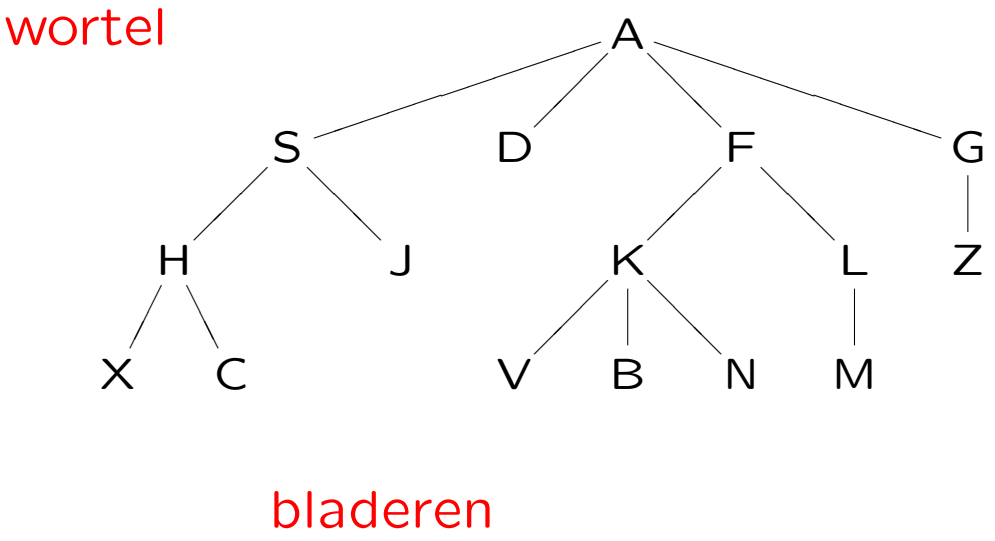
In de STL zitten overigens bijvoorbeeld ook vectoren, verzamelingen ("sets") en rijen ("queues").

Een rij (queue; denk aan een rij voor een kassa) is een reeks elementen van hetzelfde type, bijvoorbeeld karakters, met de volgende toegestane operaties:

- een lege rij aanmaken,
- testen of de rij leeg is,
- een element toevoegen (*push*),
- het eerst-toegevoegde element eruithalen (*pop*),
- soms: kijken of de rij al vol is.

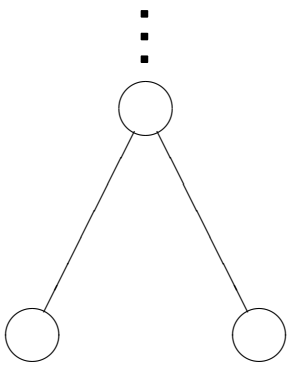
Een rij heeft dus de *FIFO-eigenschap*: FIFO = First In First Out. Toevoegen en verwijderen gebeuren dus aan verschillende kanten: *achteraan* respectievelijk *vooraan*.

Definitie: een boom is een "samenhangende ongerichte graaf zonder cykels", met één speciale knoop: de *wortel*.

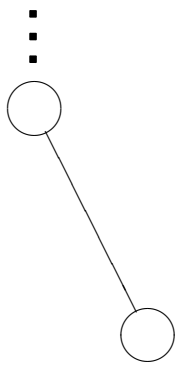


De kinderen van A zijn S, D, F en G; S is de ouder van J. Verder zijn K, L, V, B, N en M de afstammelingen van F. En H, S en A zijn de voorouders van X.

Een binaire boom is een boom waarin elke knoop ofwel nul, ofwel één, ofwel twee kinderen heeft: het *linkerkind* en het *rechterkind*. Als een knoop één kind heeft, dan is dit ofwel een linkerkind, ofwel een rechterkind.

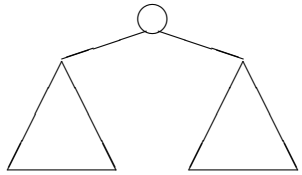


linkerkind rechterkind



één kind: rechterkind

Recursief gedefinieerd: een binaire boom
is leeg, of bestaat uit een wortel,
een linker- en een rechtersubboom.



```
class knoop {  // een struct mag ook
  public:
    knoop ( ) {  // constructor
      info = 0; links = nullptr; rechts = nullptr; }//constructor
    int info;
    knoop* links; knoop* rechts;
};//knoop

class binaireboom {
  public:
    binaireboom ( ) { wortel = nullptr; }
    void WLR ( ) { preorde (wortel); }
    void LWR ( ) { symmetrisch (wortel); }
    ...
  private:
    knoop* wortel;
    void preorde (knoop* root);
    void symmetrisch (knoop* root);
    ...
};//binaireboom
```

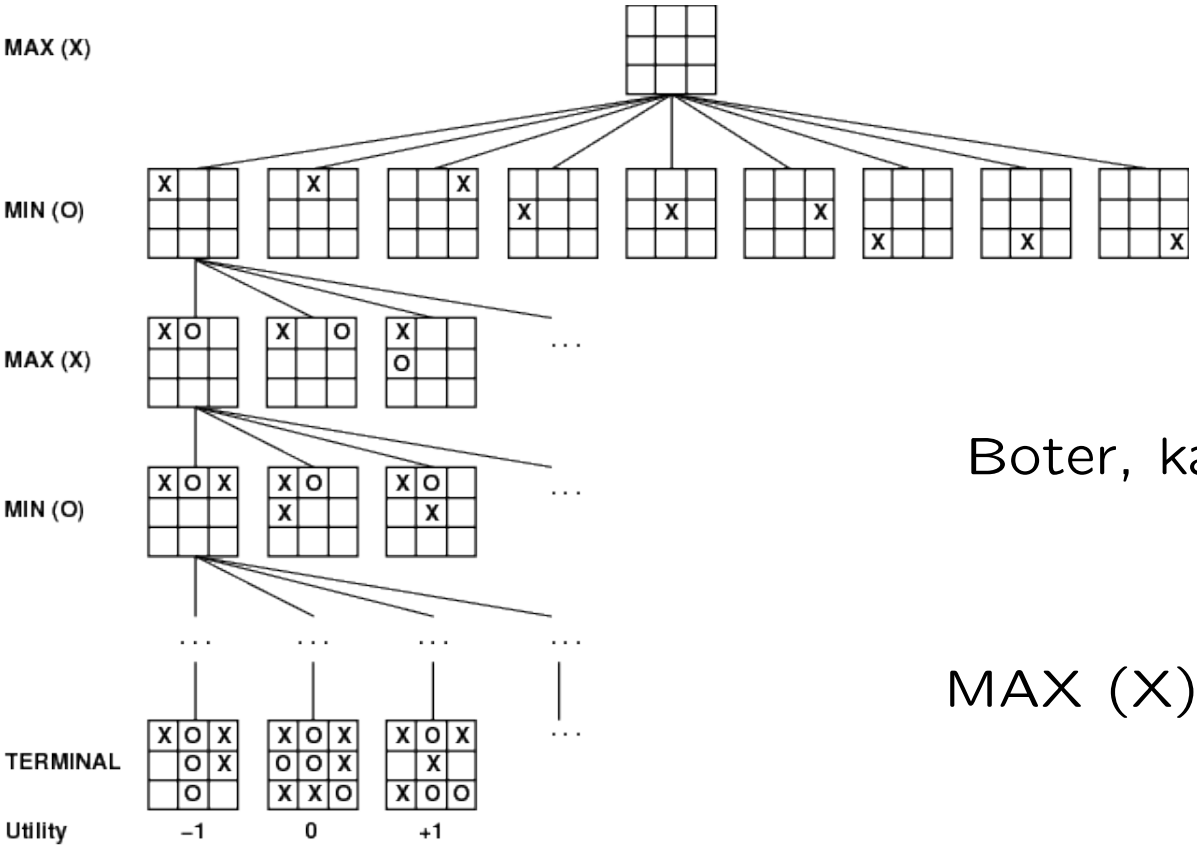
```
void binaireboom::preorde (knoop* root) {          // WLR
  if ( root != nullptr ) {          // Wortel-Links-Rechts
    cout << root->info << endl;
    preorde (root->links);
    preorde (root->rechts);
  }//if
}//preorde

void binaireboom::symmetrisch (knoop* root) {      // LWR
  if ( root != nullptr ) {          // Links-Wortel-Rechts
    symmetrisch (root->links);
    cout << root->info << endl;
    symmetrisch (root->rechts);
  }//if
}//symmetrisch
```

We tellen *recursief* het aantal knopen van een binaire boom met ingang wortel.
Aanroep: `int tellen = aantal (wortel);`

```
int aantal (knoop* root) {
  if ( root == nullptr )  // lege boom
    return 0;
  else
    return ( 1 + aantal (root->links)
               + aantal (root->rechts) );
}//aantal
```

Hier wordt eigenlijk een preorde-wandeling gedaan.

Bomen, en niet alleen binaire, worden vaak gebruikt om spellen als schaken en go te analyseren ("α-β-algoritme").



Boter, kaas en eieren

twee spelers:
MAX (X) en MIN (O)

Het *aantal vervolgpartijen* vanuit een gegeven positie in de wortel is het aantal bladeren in deze boom. Het kan berekend worden zonder de boom "echt" te maken, zie het college over recursie!

Voor Boter, kaas en eieren is dit 255.168, waarvan overigens 131.184 gewonnen door de beginspeler, 77.904 door de ander en 46.080 remise.

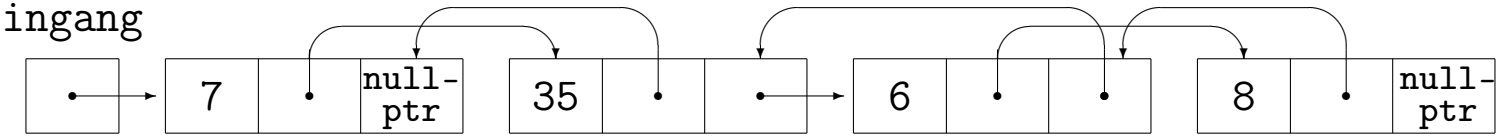
En voor zetten terugnemen kun je een stapel gebruiken: elke keer als je een zet doet, zet je de complete "oude" positie (oftewel stand) op de stapel! Soms hoef je alleen de zetten te onthouden.

Opgave 4 van het tentamen van 3 januari 2019:

Gegeven is het volgende type:

```
class object { public: int info; object* volg1; object* volg2; };
```

Met behulp hiervan kan een lijst van objecten worden opgebouwd, bestaande uit vakjes met een getal, en twee pointers. Precies een van deze twee wijst naar het volgende object, de andere naar het vorige — maar je weet niet welke. Een voorbeeld, met `ingang` van type `object*`:



a. (6) Schrijf een C++-functie `voegtoe (ingang,getal)` die een nieuw object met `getal` erin netjes vooraan de lijst met ingang `ingang` toevoegt. Je mag zelf kiezen welke van de twee pointers in het nieuwe object naar vorige en volgende object wijst. Zet in het oude eerste object (als dat bestaat) de terugwijzende pointer goed.

Opgave 4 van het tentamen van 3 januari 2019, vervolg:

b. (6) Schrijf een C++-functie `verwijder (ingang)` die het eerste object uit de lijst met ingang `ingang` netjes verwijdert, indien dit bestaat.

c. (4) Schrijf een C++-functie `hoogop (ingang)` die als er minstens twee objecten zijn en als het `info`-veld van het eerste object oneven is, dit ophoogt met het `info`-veld van het tweede object. In het voorbeeld: 7 wordt 42.

d. (3) In de functies bij **a**, **b** en **c** staat in de heading een pointer. Deze heb je call by value of call by reference doorgegeven (met een `&`). Maakt het voor de werking van deze functies verschil uit of die `&` erbij staat? Mag het, moet het? Leg duidelijk uit.

e. (6) Schrijf een C++-functie `repareer (ingang)` die ervoor zorgt dat na afloop alle `volg1`-pointers naar het volgende, en alle `volg2`-pointers naar het vorige object wijzen.

www.liacs.leidenuniv.nl/~kosterswa/pm/tentamens.php

Uitwerking Opgave 4a van 3 januari 2019:

```
void voegtoe (object* & ingang, int getal) {
  object* nieuw = new object;
  nieuw->info = getal;
  nieuw->volg1 = nullptr; // of
  nieuw->volg2 = ingang;  // andersom

  if ( ingang != nullptr )
    if ( ingang->volg1 == nullptr )
      ingang->volg1 = nieuw;
    else
      ingang->volg2 = nieuw;
  ingang = nieuw;
}//voegtoe
```

- werk aan de vierde programmeeropgave — de deadline is op maandag 13 december 2021
- lees dictaat Hoofdstuk 5
- maak opgaven 47/51 uit het opgavendictaat
- nog twee "colleges": Algoritmen en Talen: Python, oude tentamens, ...
- www.liacs.leidenuniv.nl/~kosterswa/pm/

Programmeermethoden

Algoritmen

Walter Kosters en Jonathan Vis

week 13: 6–10 december 2021

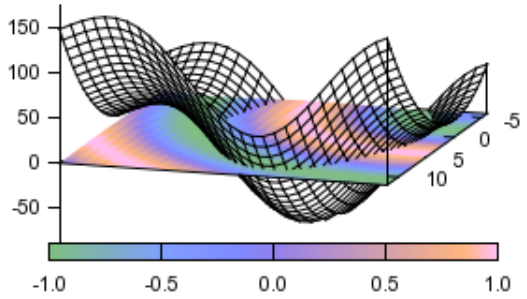
www.liacs.leidenuniv.nl/~kosterswa/pm/

Koffiesweeper programmeren we als volgt:

- week 1: pointerpracticum, opgave lezen
- week 2: klassen, pointerstructuur aanleggen, elementair spelen
- week 3: finesses, recursie, stapel
- week 4: experiment (`gnuplot`), verslag

www.liacs.leidenuniv.nl/~kosterswa/pm/op4pm.php

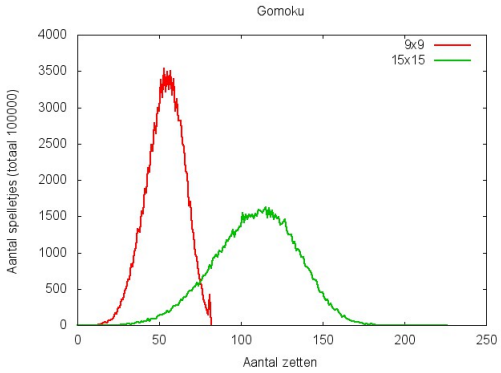
Gebruik `gnuplot` om een eenvoudige grafiek te maken, ook in Windows:




```
gnuplot> plot "stats.txt" with lines
```

Hierbij is de file `stats.txt` zoiets als:

```
1   7
2   12
3   14
```



Zie www.gnuplot.info.

Op allerlei colleges en in allerlei boeken en artikelen worden algoritmen behandeld, bijvoorbeeld bij de volgende Informatica-colleges in Leiden (semester tussen haakjes):

- Programmeermethoden (1), Algoritmiek (2)
- Datastructuren (3), Kunstmatige intelligentie (4)
- . . .

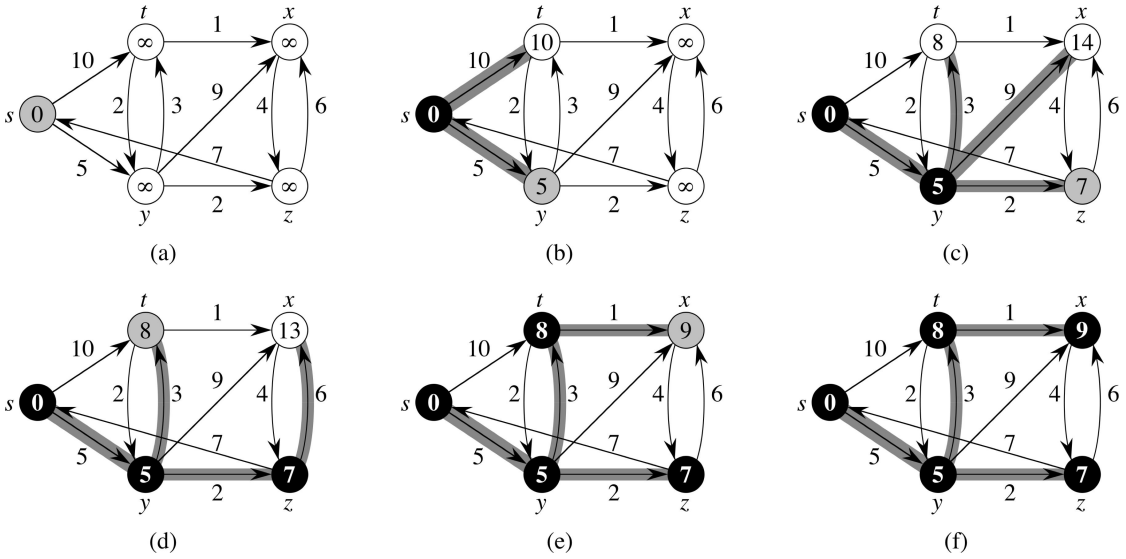
Je kunt algoritmen op allerlei manieren rubriceren, bijvoorbeeld met behulp van de volgende begrippen:

verdeel en heers, numerieke wiskunde, graafalgoritmen, dynamisch programmeren, patroonherkenning, adversary, data mining, geometrisch modelleren, backtracking, benaderende algoritmen, kunstmatige intelligentie, neurale netwerken, evolutionaire algoritmen, P↔NP, gretige algoritmen, snelle Fouriertransformatie, ...

We bekijken er een paar van.

Gegeven een graaf G, met afstanden op de takken, en twee knopen a en b. Gevraagd: kortste pad van a naar b.

Oplossing: het algoritme van Dijkstra.



Als n een priemgetal is, geldt dat $a^{n-1}-1$ deelbaar is door n voor $a = 1, 2, \ldots, n-1$. Het omgekeerde is bijna waar.

Dit suggereert het volgende algoritme dat "bepaalt" of n een priemgetal is: Als $2^{n-1}-1$ niet deelbaar is door n is n zeker geen priemgetal, en anders (misschien) wel. Dit gaat fout bij 341, 561, ..., maar dat is te verbeteren: probeer andere a; echter, 561, een Carmichael-getal, blijft lastig. (Uiteindelijk: Miller-Rabin.)

Het algoritme is een randomized algoritme, en wel een Monte Carlo algoritme: het ene antwoord is altijd juist, het anders soms niet. Bij Las Vegas algoritmen zijn de antwoorden altijd juist — maar het duurt soms lang.

En hoe maak je een willekeurige permutatie, dat wil zeggen, een random volgorde van de getallen $1, 2, \ldots, n$?

```
// stop random permutatie van 0,1,...,n-1 in array A
void maakpermutatie (int A[ ], int n) {
  int i;  // array-index
  int r;  // random array-index
  for ( i = 0; i < n; i++ ) A[i] = i;
  for ( i = n-1; i >= 0; i-- ) {
    r = rand ( ) % ( i+1 );  // 0 <= r <= i, random
    wissel (A[i],A[r]);
  }//for
}//maakpermutatie
```

`rand ( )` geeft een random-getal; `srand (42)` zet het "seed".

Een gretig (greedy) algoritme neemt beslissingen door één stap vooruit te kijken; het zijn meestal benaderende algoritmen.

We bekijken het volgende algoritme voor het Common Superstring probleem, dat vraagt naar een (zo kort mogelijke) string die een stel gegeven strings bevat: Neem herhaald de twee meest overlappende strings bij elkaar.

Een voorbeeld. Begin met `TCAGT`, `CATCAG`, `GTG` en `GCA`.
De twee meest overlappende strings zijn `CATCAG` en `TCAGT`; vervang deze door `CATCAGT`, we hebben dan `CATCAGT`, `GTG` en `GCA` over.
Zowel `GTG` als `GCA` hebben een overlap van 2 met `CATCAGT`.
Kies bijvoorbeeld `GTG`, wat `CATCAGTG` en `GCA` oplevert.
De eindoplossing is `GCATCAGTG`, en die is toevallig optimaal.

Nog een voorbeeld: begin met `GCC`, `ATGC` en `TGCAT`.
De twee meest overlappende strings zijn `ATGC` en `TGCAT`; vervang deze door `ATGCAT`, en we houden `ATGCAT` en `GCC` over.
Deze twee strings hebben geen overlap, dus de eindoplossing is hun "concatenatie": `ATGCATGCC` of `GCCATGCAT`, beide van lengte 9. De optimale oplossing, `TGCATGCC`, heeft echter lengte 8!

Het algoritme vindt wel snel een superstring, maar niet altijd een optimale . . .

Algemener: begin met

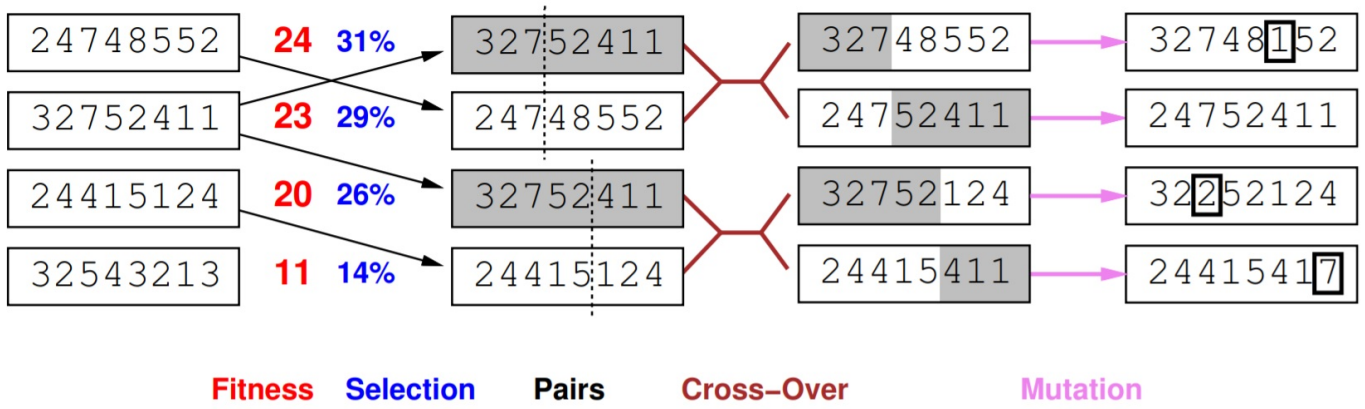
$$\{\mathtt{C}(\mathtt{AT})^k, (\mathtt{TA})^k, (\mathtt{AT})^k\mathtt{G}\}$$

voor een vaste $k \geqslant 1$, dan levert het algoritme $\mathtt{C}(\mathtt{AT})^k\mathtt{G}(\mathtt{TA})^k$ ter lengte $4k+2$, terwijl de optimale string $\mathtt{C}(\mathtt{AT})^{k+1}\mathtt{G}$ lengte $2k+4$ heeft.

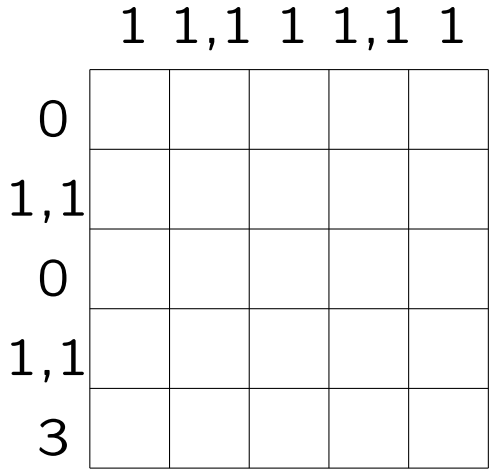
De uitvoer van het gretige algoritme kan dus twee keer zo lang zijn dan de optimale. Maar erger wordt het niet (open problemen).

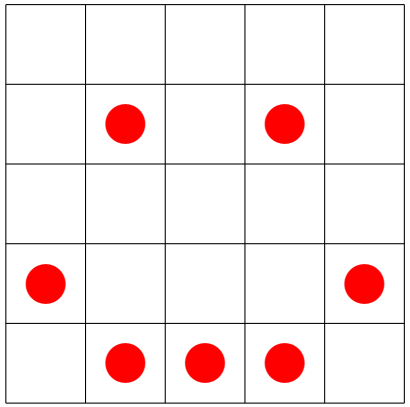
Dit soort algoritmen wordt gebruikt bij DNA-reconstructie: de "shotgun-methode".

Genetische algoritmen, of algemener Evolutionaire algoritmen, evolueren een populatie met kandidaat-oplossingen voor een probleem. Van elke kandidaat-oplossing kun je de kwaliteit berekenen met een fitness-functie. De beste individuen gaan door, en met mutatie (willekeurige, kleine veranderingen) en crossover (combineer twee "ouders") krijg je een nieuwe generatie.



Japanse puzzels (Nonogrammen) zien er zo uit:





Naast iedere rij en boven iedere kolom staan in volgorde de lengtes van aaneengesloten series rode blokjes. Zie de derde programmeeropgave van vorig jaar.

www.liacs.leidenuniv.nl/~kosterswa/nono/

Genetische algoritmen kunnen gebruikt worden om Nonogrammen op te lossen.

Een individu is hier een string (of array) met 25 bits (algemener, voor een m bij n puzzel, met mn bits), waarbij een 1 rood en een 0 leeg voorstelt. Je kunt er voor kiezen om het aantal enen per rij altijd "goed" te houden.

Mutatie zou bijvoorbeeld in een rij een 1 en een 0 kunnen verwisselen. Als je handig muteert kun je "alles" bereiken!

De fitness-functie is een som over rijen en kolommen. Per rij/kolom geef je aan hoeveel je van de specificatie afwijkt — en dat is nog lastig precies te maken.

Maar het is wellicht beter om Dynamisch Programmeren te gebruiken om Nonogrammen op te lossen.

De vraag is wat bij een lijn (rij of kolom) kan worden afgeleid, gegeven een deelinvulling:

3,2,1 | ? | ● | ? | ● | ? | • | ? | ? | ? | ? | ?

Een • betekent een zeker leeg vakje, een ● staat voor een zeker gevuld vakje. De rest is nog onbekend.

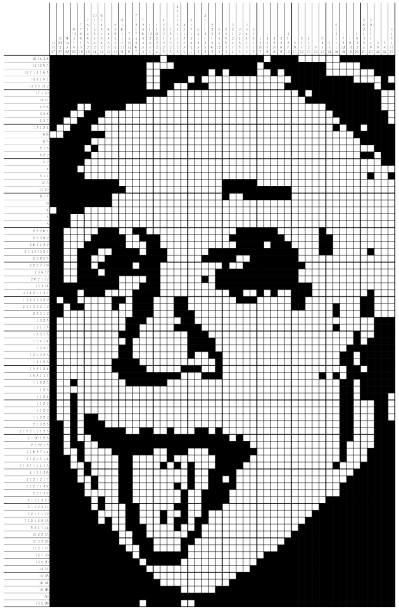
Dan concluderen we:

3,2,1 | • | ● | ● | ● | • | • | ? | ● | ? | ? | ?

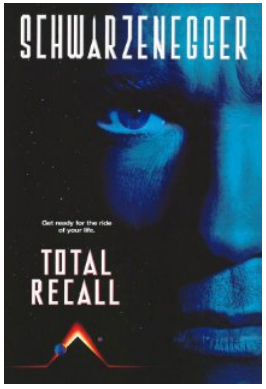
Hoe helpt Dynamisch Programmeren hierbij?

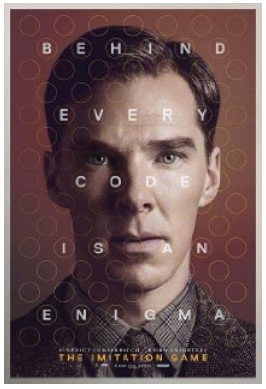
In plaats van alle manieren te bedenken waarmee de lijn kan worden gevuld (en te kijken wat deze gemeenschappelijk hebben) kun je ook tabellen maken om te zien hoe dit zit met deelrijen en deelbeschrijvingen!

We berekenen L_{ij}: kan de (deel)beschrijving $d_1, d_2, \ldots, d_i$ worden gerealiseerd in de (deel)string $s_1, s_2, \ldots, s_j$;
en zo ja: wat "moet"?
Hierbij: $0 \leqslant i \leqslant$ lengte beschrijving
en $0 \leqslant j \leqslant$ lengte string.

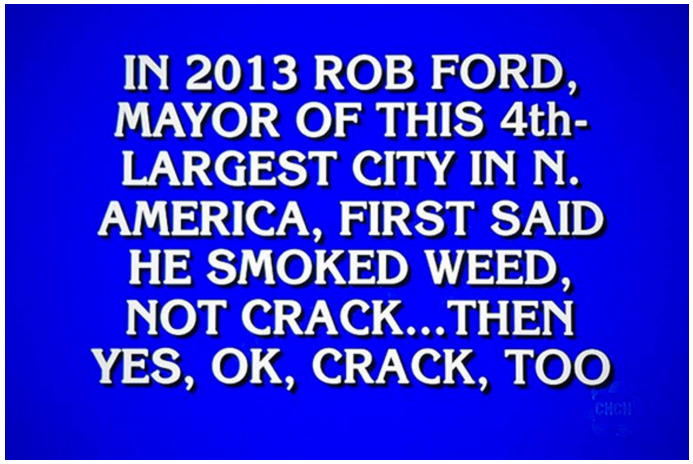


Kunstmatige intelligentie



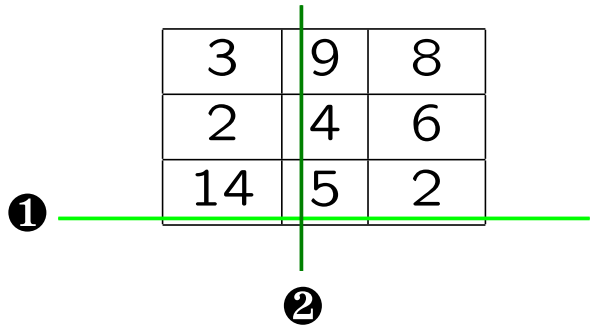


IBM: Jeopardy!



2011

Maxi en Mini spelen het volgende eenvoudige spel: Maxi wijst eerst een (horizontale) rij aan, en daarna kiest Mini een (verticale) kolom:



Bijvoorbeeld: Maxi ❶ kiest rij 3, daarna kiest Mini ❷ kolom 2; dat levert einduitslag 5.

Maxi wil graag een zo groot mogelijk getal, Mini juist een zo klein mogelijk getal.

Hoe spelen we dit spel zo goed mogelijk?

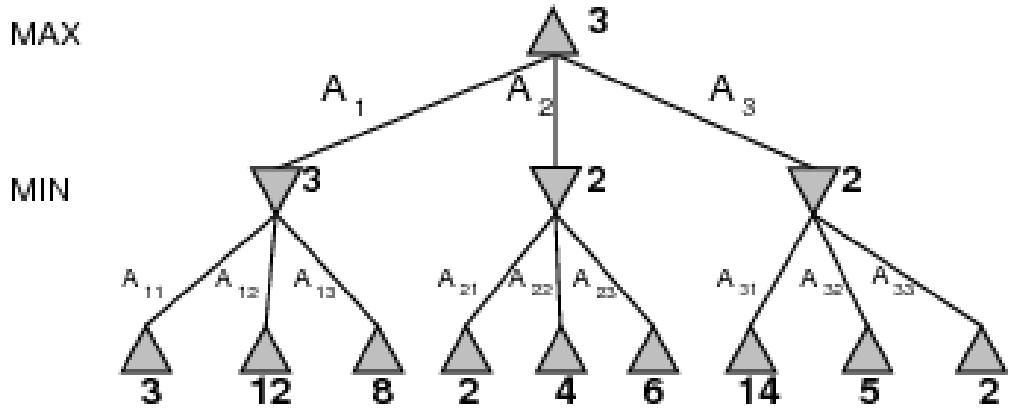
Is Maxi rij 1 kiest, kiest Mini kolom 1 (levert 3); als Maxi rij 2 kiest, kiest Mini kolom 1 (levert 2); als Maxi rij 3 kiest, kiest Mini kolom 3 (levert 2). Dus kiest Maxi rij 1!

3	9	8
2	**?**	**?**
14	5	2

Nu merken we op dat de analyse hetzelfde verloopt als we niet eens weten wat onder de twee vraagtekens zit.
Het α-β-algoritme onthoudt als het ware de beste en slechtste mogelijkheden, en kijkt niet verder als dat toch nergens meer toe kan leiden.

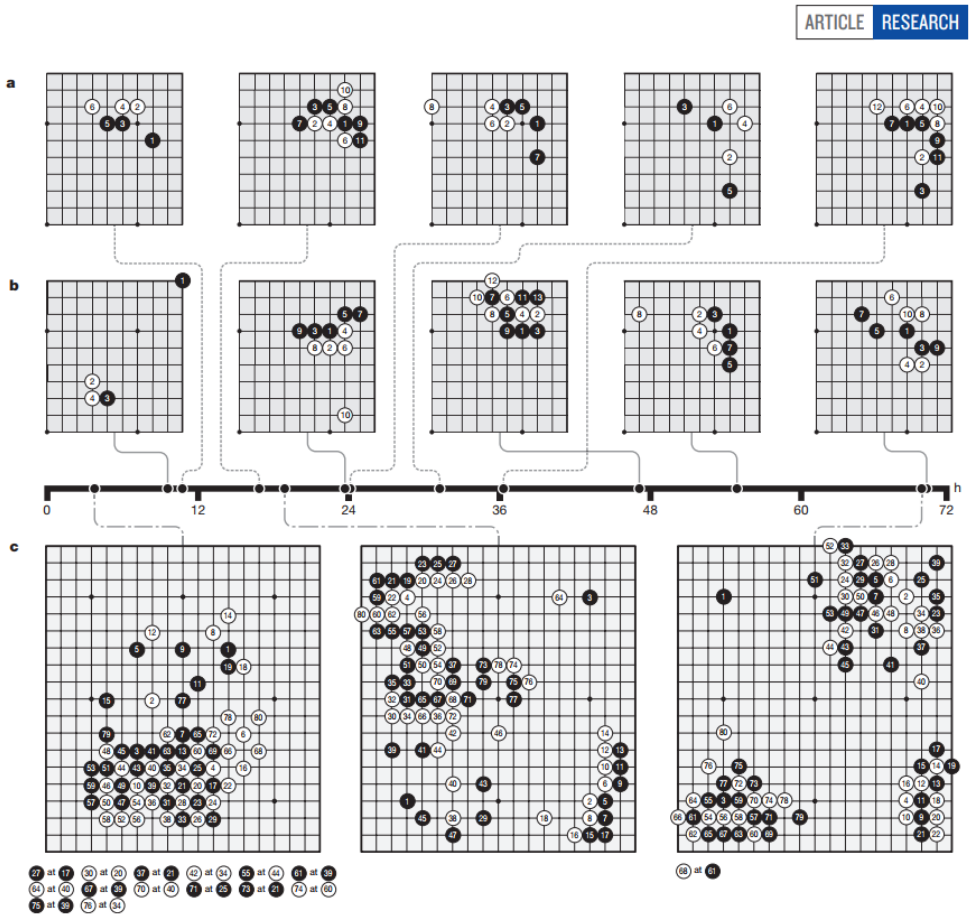
Ieder "oud" schaakprogramma gebruikt deze methode.

In boomvorm:



Het minimax-algoritme is "recursief": neem in bladeren de evaluatie-functie, in MAX-knopen het maximum van de kinderen, in MIN-knopen het minimum van de kinderen. MAX- en MIN-knopen wisselen elkaar af.

Bovenstaande boom is één zet (= move) diep, oftewel twee ply.



breaking news

↓

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver,[1*] Thomas Hubert,[1*] Julian Schrittwieser,[1*]
Ioannis Antonoglou,[1] Matthew Lai,[1] Arthur Guez,[1] Marc Lanctot,[1]
Laurent Sifre,[1] Dharshan Kumaran,[1] Thore Graepel,[1]
Timothy Lillicrap,[1] Karen Simonyan,[1] Demis Hassabis[1]

[1]DeepMind, 6 Pancras Square, London N1C 4AG.
*These authors contributed equally to this work.

Abstract

The game of chess is the most widely-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been refined by human experts over several decades. In contrast, the *AlphaGo Zero* program recently achieved superhuman performance in the game of Go, by *tabula rasa* reinforcement learning from games of self-play. In this paper, we generalise this approach into a single *AlphaZero* algorithm that can achieve, *tabula rasa*, superhuman performance in many challenging domains. Starting from random play, and given no domain knowledge except the game rules, *AlphaZero* achieved within 24 hours a superhuman level of play in the games of chess and shogi (Japanese chess) as well as Go, and convincingly defeated a world-champion program in each case.

The study of computer chess is as old as computer science itself. Babbage, Turing, Shannon, and von Neumann devised hardware, algorithms and theory to analyse and play the game of chess. Chess subsequently became the grand challenge task for a generation of artificial intelligence researchers, culminating in high-performance computer chess programs that perform at superhuman level (*9, 13*). However, these systems are highly tuned to their domain, and cannot be generalised to other problems without significant human effort.

A long-standing ambition of artificial intelligence has been to create programs that can instead learn for themselves from first principles (*26*). Recently, the *AlphaGo Zero* algorithm achieved superhuman performance in the game of Go, by representing Go knowledge using deep convolutional neural networks (*22, 28*), trained solely by reinforcement learning from games of self-play (*29*). In this paper, we apply a similar but fully generic algorithm, which we

Silver et al., Nature 550 (2017) 354–359: Deep learning, TPU's, . . .

Nature, oktober 2019: breaking news

Article

Grandmaster level in StarCraft II using multi-agent reinforcement learning

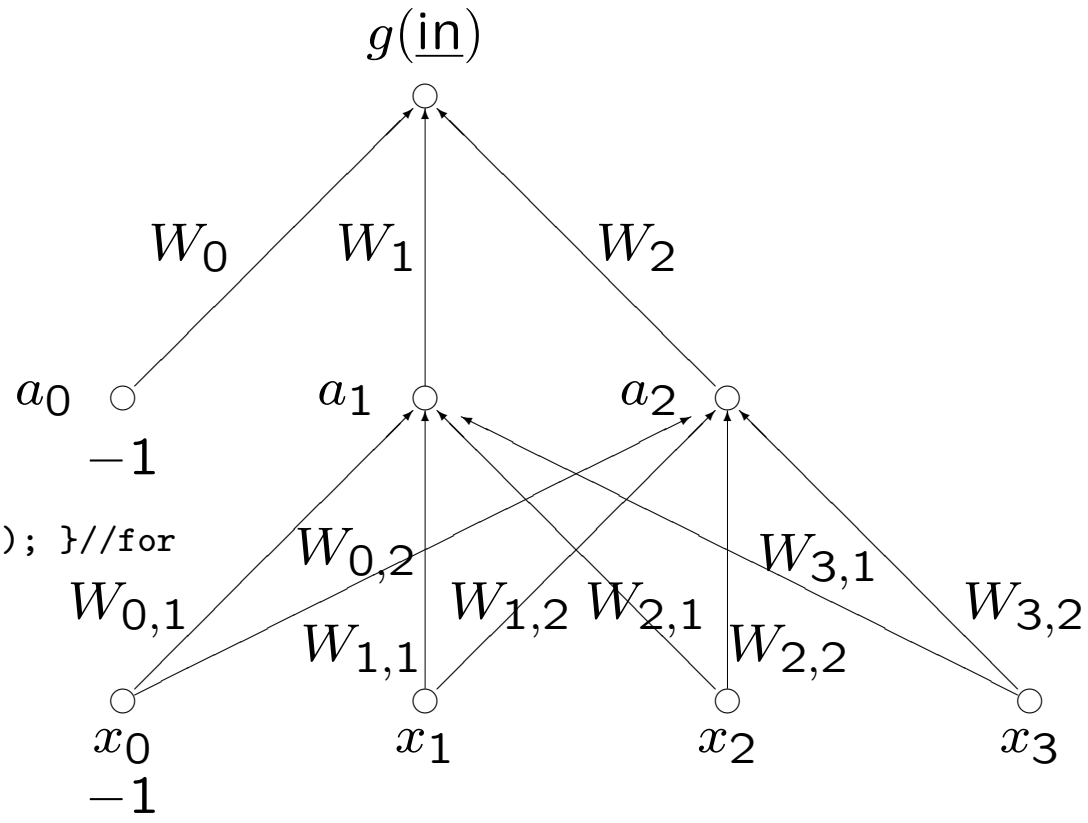
https://doi.org/10.1038/s41586-019-1724-z
Received: 30 August 2019
Accepted: 10 October 2019
Published online: 30 October 2019

Oriol Vinyals[1,3]*, Igor Babuschkin[1,3], Wojciech M. Czarnecki[1,3], Michaël Mathieu[1,3], Andrew Dudzik[1,3], Junyoung Chung[1,3], David H. Choi[1,3], Richard Powell[1,3], Timo Ewalds[1,3], Petko Georgiev[1,3], Junhyuk Oh[1,3], Dan Horgan[1,3], Manuel Kroiss[1,3], Ivo Danihelka[1,3], Aja Huang[1,3], Laurent Sifre[1,3], Trevor Cai[1,3], John P. Agapiou[1,3], Max Jaderberg[1], Alexander S. Vezhnevets[1], Rémi Leblond[1], Tobias Pohlen[1], Valentin Dalibard[1], David Budden[1], Yury Sulsky[1], James Molloy[1], Tom L. Paine[1], Caglar Gulcehre[1], Ziyu Wang[1], Tobias Pfaff[1], Yuhuai Wu[1], Roman Ring[1], Dani Yogatama[1], Dario Wünsch[2], Katrina McKinney[1], Oliver Smith[1], Tom Schaul[1], Timothy Lillicrap[1], Koray Kavukcuoglu[1], Demis Hassabis[1], Chris Apps[1,3] & David Silver[1,3]*

Many real-world applications require artificial agents to compete and coordinate with other agents in complex environments. As a stepping stone to this goal, the domain of StarCraft has emerged as an important challenge for artificial intelligence research, owing to its iconic and enduring status among the most difficult professional esports and its relevance to the real world in terms of its raw complexity and multi-agent challenges. Over the course of a decade and numerous competitions[1–3], the strongest agents have simplified important aspects of the game, utilized superhuman capabilities, or employed hand-crafted sub-systems[4]. Despite these advantages, no previous agent has come close to matching the overall skill of top StarCraft players. We chose to address the challenge of StarCraft using general-purpose learning methods that are in principle applicable to other complex domains: a multi-agent reinforcement learning algorithm that uses data from both human and agent games within a diverse league of continually adapting strategies and counter-strategies, each represented by deep neural networks[5,6]. We evaluated our agent, AlphaStar, in the full game of StarCraft II, through a series of online games against human players. AlphaStar was rated at Grandmaster level for all three StarCraft races and above 99.8% of officially ranked human players.

Kunnen computers denken?

```
for ( j = 1; j <= hs; j++ ) {
  s[j] = -ItH[0][j];
  for ( k = 1; k <= ip; k++ )
    s[j] += ItH[k][j] * I[k];
  HtA[j] = g (s[j]); }//for
for ( i = 0; i < os; i++ ) {
  sO = -HtO[0][i];
  for ( j = 1; j <= hs; j++ )
    sO += HtO[j][i] * HtA[j];
  O[i] = g (sO);
  dO[i] = gp (sO) * ( T[i] - O[i] ); }//for
for ( j = 1; j <= hs; j++ ) {
  d[j] = 0;
  for ( i = 0; i < os; i++ )
    d[j] += HtO[j][i] * dO[i];
  d[j] *= gp (sum[j]); }//for
for ( j = 0; j <= hs; j++ )
  for ( i = 0; i < os; i++ )
    HtO[j][i] += a * HtA[j] * dO[i];
for ( k = 0; k <= ip; k++ )
  for ( j = 1; j <= hs; j++ )
    ItH[k][j] += a * I[k] * d[j];
```

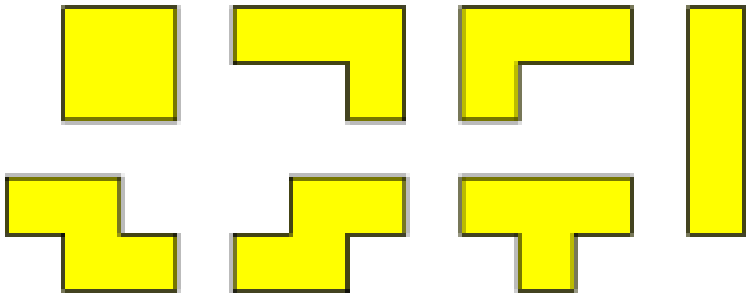


Aan een spel als Tetris kleven allerlei vragen:

- Hoe speel je het zo goed mogelijk?
  (AI = Kunstmatige intelligentie)
- Hoe moeilijk is het? (complexiteit)
- Wat kan er allemaal gebeuren?

Zo is bijvoorbeeld bewezen dat sommige Tetris-vragen NP-volledig zijn (gezamenlijk werk met mensen van MIT), dat je bijna alle configuraties kunt bereiken, maar dat niet alle problemen "beslisbaar" zijn.

De 7 Tetris-stukken:



De vraag "Kun je met een gegeven serie (inclusief volgorde) van deze stukken een deels al gevuld bord helemaal leeg spelen?" is NP-volledig.
Als iemand het bord leeg speelt kun je dat eenvoudig controleren. Als het *niet* kan, kan men (tot nu toe) niks beters verzinnen dan alle mogelijkheden één voor één na te gaan!

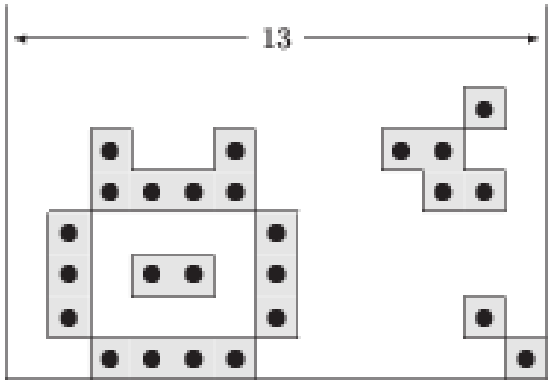
P en NP zijn "klassen" van beslissingsproblemen. Het probleem of een graaf samenhangend is, zit in P. Het probleem of een graaf een Hamilton-circuit heeft, zit in NP, en is zelfs NP-volledig; je kunt het "eenvoudig", maar niet efficiënt, oplossen met bruteforce technieken.

P is de klasse van beslissingsproblemen die door een "deterministische Turing-machine" in "polynomiale tijd" kunnen worden opgelost. NP is de klasse van beslissingsproblemen die door een "niet-deterministische Turing-machine" in "polynomiale tijd" kunnen worden opgelost: je mag "gokken".

Open probleem: geldt P = NP?

Tetris: bereikbaar?

Een "willekeurige" configuratie:



Deze kan gemaakt worden door 276 geschikte Tetris-stukken op de juiste plaats te laten vallen.

Claim: op een bord van oneven breedte kan elke configuratie bereikt worden!

www.liacs.leidenuniv.nl/~kosterswa/tetris/

Opgave 1b van het tentamen van 6 januari 2020:

In het array `int A[n]` staan `n` (een `const int` $\geqslant 3$) verschillende gehele getallen.
Schrijf een C++-functie `int stijg (A,b,n)` die de lengte van een langste stijgende aaneengesloten deelrij van `A` geeft, en diens begin-index in `b` oplevert. Als er meerdere deelrijen het langste zijn: de kleinste `b`. Dus array 2 7 4 5 6 1 3 8 geeft 3, met `b` = 2 (deelrij 4 5 6, even lang als 1 3 8).

```
b. int stijg (int A[ ], int & b, int n ) {
     int i, langste = 1, lang = 1, begin = 0; b = 0;
     for ( i = 1; i < n; i++ )
       if ( A[i] > A[i-1] ) {
         lang++;
         if ( lang > langste ) { langste = lang; b = begin; }//if
       }//if
       else {
         begin = i; lang = 1;
       }//else
     return langste;
   }//stijg
```

Opgave 2 van het tentamen van 6 januari 2015:

a. Bij een functie kun je te maken hebben met *call by value* en *call by reference*, en ook met *locale* en *globale* variabelen. Verder onderscheiden we ook nog *formele* en *actuele* parameters. Leg deze zes begrippen duidelijk uit.

b. Gegeven een C++-programma met daarin de volgende twee functies:

```
int ludo (int a, int b, int n) {
  int i = 42; for ( i = 0; i < n; i += 2 ) { b += a; i--; }//for
  return b; }//ludo
int jeanine (int a, int b) {
  a = ludo (a,b,a); cout << a << "," << b << endl;
  a = ludo (a,b,a); cout << a << "," << b << endl;
  return a; }//jeanine
```

Verder zijn de globale variabelen `x` en `y` gegeven (van type `int`). Wat is dan de uitvoer van het volgende stukje programma (leg je antwoord duidelijk uit):

```
x = 2; y = 2; y = jeanine (x,y); cout << x << "," << y << endl;
```

c. Als **b**, maar nu met een `&` (“ampersand”) bij de vijf parameters van de functies.

d. Geef een eenvoudige uitdrukking voor de return-waarde van een aanroep `ludo (a,b,n)`, uitgedrukt in diens parameters. Het maakt niet uit of er `&`’s bij de parameters staan.

e. Als **d**, maar nu voor `jeanine (a,b)`. Neem aan dat er bij alle vijf parameters een `&` staat, net als bij **c**.

f. Als in de functie `ludo` ergens `a = jeanine (a,2*n);` staat, compileert het programma dan nog? Onderscheid gevallen met en zonder `&`.

- werk aan de vierde programmeeropgave — de deadline is op maandag 13 december 2021

- volgende week laatste college, over Talen als Python & oude tentamens

- www.liacs.leidenuniv.nl/~kosterswa/pm/

Programmeermethoden

Talen als Python; oude tentamens

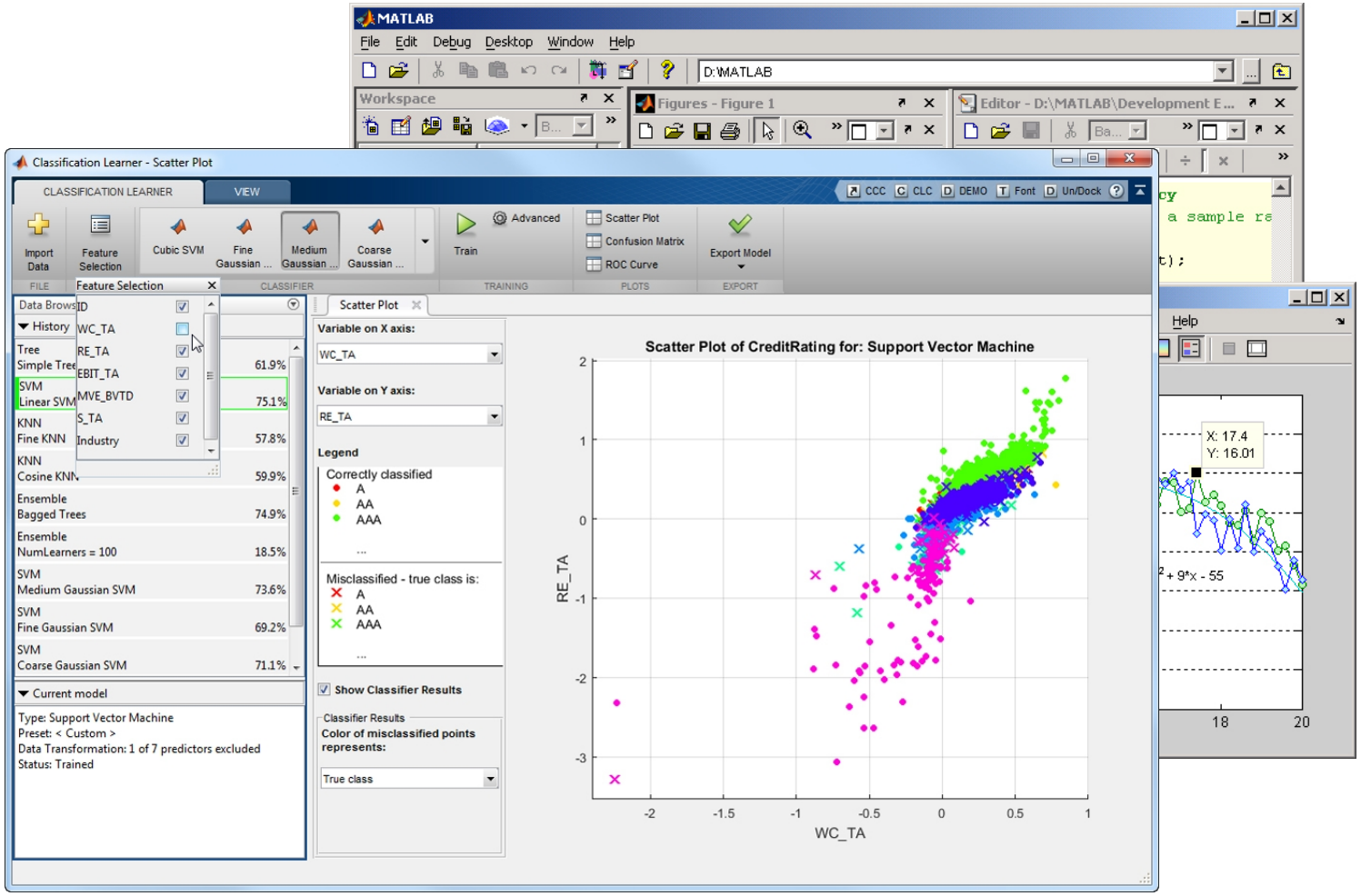
Walter Kosters en Jonathan Vis

week 14: 13–17 december 2021

www.liacs.leidenuniv.nl/~kosterswa/pm/

Naast C++ (C: Brian Kernighan, Dennis Ritchie, Ken Thompson, $\pm$ 1970; C++: Bjarne Stroustrup, $\pm$ 1985, nu C++20), wordt ook veel MATLAB gebruikt, en Java. En Qt voor interfaces. En . . . Python.

Ritchie en Thompson in 1972



www.mathworks.com

MATLAB, zelf in C geschreven, is met name goed in allerlei wiskundige operaties: matrices, . . . ; de taal heeft ook veel grafische mogelijkheden: plotten, . . .
Nadeel: MATLAB is duur.

Naast de workspace heb je de "command history" en de "command window".
Programma's (scripts) hebben extensie `.m`.

MATLAB is, net als Python, "weakly dynamically typed".

De programmeertaal Java lijkt veel op C++.

Enkele belangrijke verschillen:

- Java is (nog) meer object-georienteerd
- Java heeft automatische garbage collection
- Java heeft (bijna) geen pointers
- Java is platform-onafhankelijk
- Java kent applets: WWW-applicaties, met GUI's

Hoe compileer je een Java-programma, dat in `Iets.java` staat? Met `javac *.java` compileer je al je Java-files naar bytecode, in ons geval naar `Iets.class`. Deze kan met `java Iets` op elke computer gedraaid (om precies te zijn: *geinterpreteerd*) worden met behulp van de Java Virtual Machine (JVM). Dit bovenstaande geldt voor "tekst-applicaties".

Een applet wordt bekeken met een webbrowser, en je moet dan een aparte HTML-pagina `naam.html` maken die de bytecode aanroept. Rechtstreeks (zonder browser) kan vaak ook eenvoudig met `appletviewer naam.html`, zie later.

Overigens, Java is iets heel anders dan JavaScript.

Maak een file `Hello.java` met

```
public class Hello {
  public static void main (String[ ] argv) {
    System.out.println ("Hello world!");
  }//main
}//class Hello
```

En dan `javac Hello.java` en `java Hello`.

Let op: filenaam = naam `public` klasse (met hoofdletter).

Er kunnen meer klasses in een file staan, maar er is er maar één `public`, en bij diens `public static void main` begint de executie van het programma.

```
import java.awt.*; import java.applet.*; import java.awt.event.*;
public class Muis extends Applet //Swing
       implements MouseListener, MouseMotionListener {
  public int teller = 0;
  public void init ( ) {
    addMouseListener (this); addMouseMotionListener (this); }//init
  public void paint (Graphics g) {
    g.setColor(Color.white); g.fillRect(10,10,500,500); }//paint
  public void mouseClicked (MouseEvent event) { }//mouseClicked
  public void mouseReleased (MouseEvent event) { }//mouseReleased
  public void mousePressed (MouseEvent event) { }//mousePressed
  public void mouseEntered (MouseEvent event) { }//mouseEntered
  public void mouseExited (MouseEvent event) { }//mouseExited
  public void mouseDragged (MouseEvent event) {
    Graphics g = getGraphics ( ); teller++;
    switch ( teller % 4 ) {
      case 0: g.setColor (Color.red); break;
      case 1: g.setColor (Color.green); break;
      case 2: g.setColor (Color.black); break;
      case 3: g.setColor (Color.blue); break; }//switch
    g.drawLine (event.getX ( ),event.getY ( ),
                event.getX ( )+10,event.getY ( )+10); }//mouseDragged
  public void mouseMoved (MouseEvent event) { }//mouseMoved
}//Muis
```

www.liacs.leidenuniv.nl/~kosterswa/java/muis.html

www.liacs.leidenuniv.nl/~kosterswa/java/sleep.html

Wat we hier —en straks ook in Qt— zien is dat de "control-flow" event-driven (event-gestuurd) is. En dat is anders dan wat we gewend zijn!

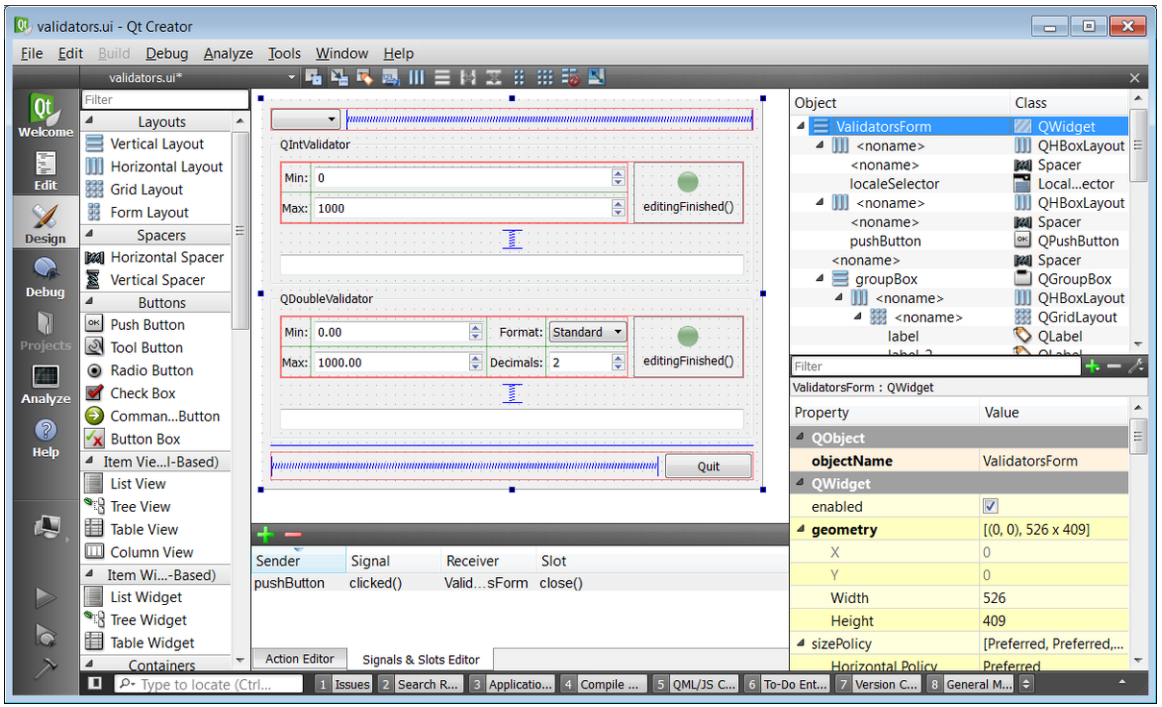
Een onzichtbare hoofdloop (de "event-lus") loopt "eindeloos" door. Intussen worden, door events zoals muis-acties af te handelen, de gebruikerswensen vervuld. Dat zijn precies methoden (= memberfuncties) behorend bij bepaalde objecten.

Qt (“cute” of “ku-tee”; de ’t’ komt van toolkit) is een “cross-platform ontwikkelomgeving”, met een ingebouwde C++-bibliotheek. Samenwerken met Java kan ook. Voor niet commercieel gebruik is het gratis.

KDE, Google Earth, Mathematica en VLC zijn erin/mee geschreven.

Qt werkt met signals en slots. Als je iets doet, geeft je een signal, dat ergens door een slot wordt opgevangen.

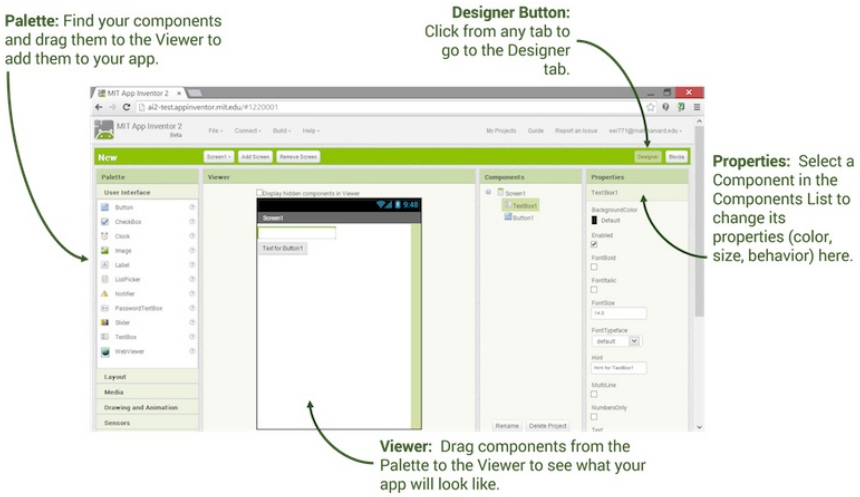
Hoe leer je Qt? Of Qt Designer?
Ga naar een Linux-systeem. Geef in een terminal-window het commando `designer &` om Qt Designer te starten.

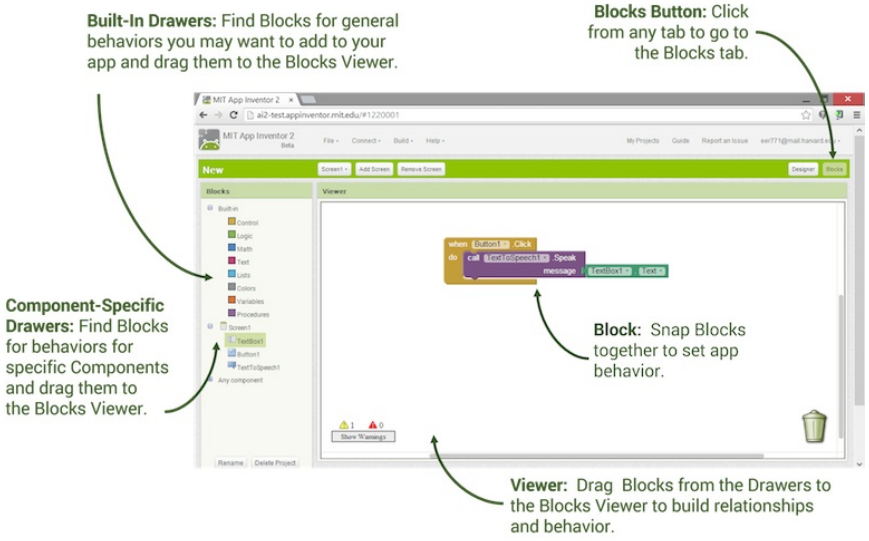


Maak vervolgens een mooie GUI (Grafische User Interface) voor Boter, kaas en eieren. Gebruik daarbij de C++-code in `boter.cc` die hier te vinden is:

www.liacs.leidenuniv.nl/~kosterswa/pm/qt.php

En hoe maak je snel een app? Bijvoorbeeld met MIT APP Inventor, zie appinventor.mit.edu/





Python, ook zeer geschikt voor scripting en "prototyping", is ontwikkeld door Guido van Rossum uit Nederland. Er zit veel overlap in met de andere genoemde talen.

Voor meer informatie (zelfstudie), zie:

www.liacs.leidenuniv.nl/~kosterswa/pm/pythonextra.php

- interpreteren, niet compileren: `python hello.py`

```
def hello():
    print("Hello world!")

hello()
```

- interactieve mode
- type van variabele kan eenvoudig wijzigen
- indentatie
- NumPy, Matplotlib, . . .

```
# Dit is een regel met commentaar
import math # voor "pi"
print("Geef straal, en Enter .. ",end="")
straal = float(input())
if straal > 0:
    print("Oppervlakte: ",end="")
    print(math.pi * straal * straal)
else:
    print("Niet zo negatief ...")
print("Einde van dit programma.")
exit(0)
```

Geen accolades, geen punt-komma's. Dit is overigens Python3; er zijn subtiele verschillen met oudere versies.

Interactieve mode, via `python3`:

```
>>> a, b, c = 1, 7.1234, "een string" # gebruik " of '
>>> print(type(a),type(b),type(c))
<class 'int'> <class 'float'> <class 'str'>
>>> a = "wat anders"
>>> print(type(a),a)
<type 'str'> 'wat anders'
>>> 42**97  # machtsverheffen
285220783529230204153175149217567486191944890823858197365940414630662077361378060803659240001284703557970677097414100792138630484270254055597330734645457715 2L
>>> a+d  # gaat fout:
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'd' is not defined
>>> Ctrl-D
```

Let op de indentatie, oftewel het inspringen:

```
if y >= 3 and ( x == 4 or x == 5 ): # and voor && ...
    pass # doe niets
elif z == 12:  # in plaats van else if uit C++
    x = 0
    y = 678
else:
    print("Hoe verzin je het!")


for karakter in ['a', 'e', 'i', 'o', 'u']:
    print(karakter,end="")
som = 0
for i in range(6):  # i = 0,1,2,3,4,5
    som = som + i*i
```

```
def telop(a,b):
    c = a + b
    return c

q = telop(12345,6789)


f = open("mijnfile.txt","r")
for regel in f:
    print(regel,end="")
f.close()
```

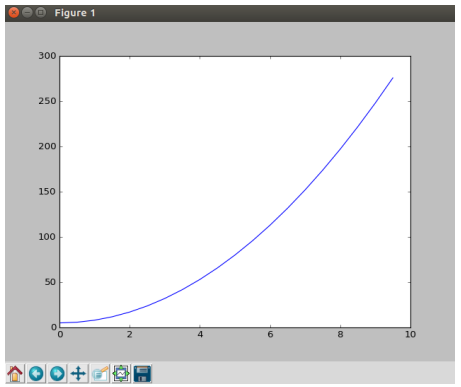
Parameter-overdracht in Python is call-by-object-reference:

```
def vergroot(lijst):
    lijst += [10,20,30,40]

def vernieuw(lijst):
    lijst = [1000,1001]

lijst = [7,8]   # een "mutable" object
vergroot(lijst)
# nu is lijst [7,8,10,20,30,40]
vernieuw(lijst)
# en nu is lijst nog steeds [7,8,10,20,30,40]
```

En integers zijn “immutable”.

```
def simpelsort(A):
    for i in range(len(A)):
    # Zoek kleinste element in ongesorteerde stuk [i:]
        kl = i
        for j, el in enumerate(A[i:]):
            if el < A[kl]:
                kl = i + j
        # Wissel om
        if i != kl:
            A[i], A[kl] = A[kl], A[i]

# Test
B = [47, 10, 7, 3, 31, 75, 18, 21, 48, 79]
simpelsort(B)
print(B)
```

```
import numpy as np  # hier zitten ook arrays in!
import matplotlib.pyplot as plt

# Bepaal de x-coordinaten waarvoor we willen plotten
x = np.arange(0, 10, 0.5)
# Bereken nu voor elk x-coordinaat de y-waarde
# Functie: y = 3x^2 + 5
y = 3 * x * x + 5

# Geef de x- en y-arrays als parameters aan de plot-functie
plt.plot(x, y)
# Zet de plot op het scherm
plt.show()
exit(0)
```



Opgave 1 van het tentamen van 6 januari 2014:

In een array `int A[n]` staan `n` (een `const` > 0) gehele getallen.

a. Schrijf een C++-functie `hoevaak (A,X,n)` die teruggeeft hoe vaak het gehele getal `X` in het array `A` voorkomt.

b. Schrijf een Booleaanse C++-functie `uniek (A,n)` die precies dan `true` teruggeeft als geen enkel getal twee maal (of vaker) voorkomt in `A`, en anders `false`. Hierbij moet de functie van **a** *zinvol* gebruikt worden (hoe vaak komt `A[i]` voor?).

c. Schrijf een C++-functie `meest (A,n)` die het meest voorkomende getal uit `A` teruggeeft. Als er verschillende kandidaten zijn (bijvoorbeeld voor het array `17 12 30 12 42 30`) moet het kleinste getal dat het meest voorkomt worden geretourneerd. In het voorbeeld is dit `12` (dat even vaak voorkomt als `30`). Maak opnieuw gebruik van de functie van **a**.

d. Schrijf een C++-functie `sorteer (A,n)` die de getallen in `A` zodanig ordent dat voor alle getallen (behalve het laatste) geldt dat ze hooguit even vaak voorkomen als hun rechter buurman. Tip: pas de C++-code voor *bubblesort* eenvoudig aan; gebruik **a**.

e. Hoe vaak wordt de functie `hoevaak` aangeroepen in **d**?

soms dus ook iets over "complexiteit": $O(n^2)$ ...

Uitwerking Opgave 1

```
a. int hoevaak (int A[ ], int X, int n) {
     int i, teller = 0;
     for ( i = 0; i < n; i++ ) if ( X == A[i] ) teller++;
     return teller;
   }//hoevaak
b. bool uniek (int A[ ], int n) {
     int i;
     for ( i = 0; i < n; i++ )
       if ( hoevaak (A,A[i],n) > 1 ) return false;
     return true;
   }//uniek
c. int meest (int A[ ], int n) {
     int i, tel, vaak = A[0], aantal = hoevaak (A,A[0],n);
     for ( i = 1; i < n; i++ ) {
       tel = hoevaak (A,A[i],n);
       if ( tel > aantal || ( tel == aantal && A[i] < vaak ) ) {
         vaak = A[i]; aantal = tel; }//if
     }//for
     return vaak;
   }//meest
d. void sorteer (int A[ ], int n) {
     int i, j, temp;
     for ( i = 1; i < n; i++ )
       for ( j = 0; j < n-i; j++ )
         if ( hoevaak (A,A[j],n) > hoevaak (A,A[j+1],n) ) {
           temp = A[j]; A[j] = A[j+1]; A[j+1] = temp; }//if
   }//sorteer
e. 2 ( 1+2+...+n-1) = n (n-1) keer
```

Opgave 2 van het tentamen van 6 januari 2015:

a. Bij een functie kun je te maken hebben met *call by value* en *call by reference*, en ook met *locale* en *globale* variabelen. Verder onderscheiden we ook nog *formele* en *actuele* parameters. Leg deze zes begrippen duidelijk uit.

b. Gegeven een C++-programma met daarin de volgende twee functies:

```
int ludo (int a, int b, int n) {
  int i = 42; for ( i = 0; i < n; i += 2 ) { b += a; i--; }//for
  return b; }//ludo
int jeanine (int a, int b) {
  a = ludo (a,b,a); cout << a << "," << b << endl;
  a = ludo (a,b,a); cout << a << "," << b << endl;
  return a; }//jeanine
```

Verder zijn de globale variabelen `x` en `y` gegeven (van type `int`). Wat is dan de uitvoer van het volgende stukje programma (leg je antwoord duidelijk uit):

```
x = 2; y = 2; y = jeanine (x,y); cout << x << "," << y << endl;
```

c. Als **b**, maar nu met een `&` (“ampersand”) bij de vijf parameters van de functies.

d. Geef een eenvoudige uitdrukking voor de return-waarde van een aanroep `ludo (a,b,n)`, uitgedrukt in diens parameters. Het maakt niet uit of er `&`’s bij de parameters staan.

e. Als **d**, maar nu voor `jeanine (a,b)`. Neem aan dat er bij alle vijf parameters een `&` staat, net als bij **c**.

f. Als in de functie `ludo` ergens `a = jeanine (a,2*n);` staat, compileert het programma dan nog? Onderscheid gevallen met en zonder `&`.

Uitwerking Opgave 2b

"main"		"jeanine"		"ludo"				
x	y	a	b	a	b	n	i	
2	2							
		2	2					aanroep `jeanine`
				2	2	2	42	eerste aanroep `ludo`
							0	
					4		1	
					6		2	geeft 6 terug
		6						`cout`: 6,2
				6	2	6	42	tweede aanroep `ludo`
							0	
					8		1	
					⋮		⋮	
					38		6	geeft 38 terug
		38						`cout`: 38,2; geeft 38 terug
	38							`cout`: 2,38

tijd ↓

Uitwerking Opgave 2c

"main"		"ludo"	tijd ↓
x = a uit jeanine = a uit ludo = n uit ludo	y = b uit jeanine = b uit ludo	i	
2	2		
			aanroep jeanine
		42	eerste aanroep ludo
		0	
	4	1	
	6	2	geeft 6 terug
6			cout: 6,6
		42	tweede aanroep ludo
		0	
	12	1	
	⋮	⋮	
	42	6	geeft 42 terug
42			cout: 42,42; geeft 42 terug
	42		cout: 42,42

d. $\mathtt{n} \cdot \mathtt{a} + \mathtt{b}$ (als $\mathtt{n} \geqslant 0$) **e**. $(\mathtt{a}^2 + \mathtt{b})^2 + \mathtt{a}^2 + \mathtt{b}$

f. Dan moet de tweede parameter van `jeanine` call by value zijn, en een prototype van `jeanine` moet boven `ludo` staan.

Opgave 3 van het tentamen van 16 maart 2017:

Gegeven is het twee-dimensionale array `int afstand[n][n];`, met zekere `const int n` $\geqslant 2$. Er geldt dat `afstand[i][j]` > 0 de afstand is tussen de plaatsen `i` en `j` met `i` $\neq$ `j`, waarbij `afstand[i][i]` `0` is, en de afstand tussen `i` en `j` even groot is als die tussen `j` en `i`. Een voorbeeld met `n = 4` staat hiernaast.

0	3	7	9
3	0	4	14
7	4	0	8
9	14	8	0

a. Schrijf een C++-functie `bool reis (afstand,km)` die kijkt of er een rondreis van `i` naar `j` naar `k` naar `i` is (voor willekeurige onderling verschillende plaatsen `i`, `j` en `k`) die in totaal precies afstand `km` heeft. In het voorbeeld zou voor `26` het antwoord `true` zijn: van `0` naar `1` naar `3` naar `0` (dat is `3 + 14 + 9` $=$ `26`).

b. Schrijf een C++-functie `int verste (afstand,i)` die het nummer van de verst van `i` afgelegen plaats oplevert. In het voorbeeld, met `i` $=$ `3`, is dat `1` (wegens afstand `14`). Als er meerdere plaatsen voldoen: die met de grootste index. Neem aan dat `0` $\leqslant$ `i` $<$ `n`.

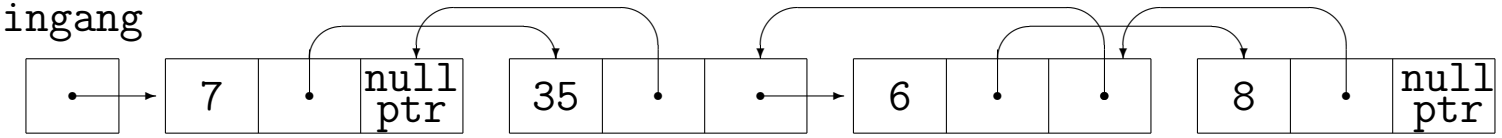
c. Schrijf een C++-functie `int hoever (afstand,i)` die bepaalt hoeveel je in totaal reist als je begint in `i`, dan steeds naar de verst gelegen plaats gaat, en stopt zodra je ergens komt waar je al eerder geweest was. In het voorbeeld, met `i` $=$ `0`, is het antwoord `9 + 14 + 14` $=$ `37` (van `0` naar `3` naar `1` naar `3`). Neem aan dat `0` $\leqslant$ `i` $<$ `n`. Hint: gebruik een Booleaans hulparray.

Opgave 4 van het tentamen van 3 januari 2019:

Gegeven is het volgende type:

```
class object { public: int info; object* volg1; object* volg2; };
```

Met behulp hiervan kan een lijst van objecten worden opgebouwd, bestaande uit vakjes met een getal, en twee pointers. Precies een van deze twee wijst naar het volgende object, de andere naar het vorige — maar je weet niet welke. Een voorbeeld, met `ingang` van type `object*`:



a. (6) Schrijf een C++-functie `voegtoe (ingang,getal)` die een nieuw object met `getal` erin netjes vooraan de lijst met ingang `ingang` toevoegt. Je mag zelf kiezen welke van de twee pointers in het nieuwe object naar vorige en volgende object wijst. Zet in het oude eerste object (als dat bestaat) de terugwijzende pointer goed.

Opgave 4 van het tentamen van 3 januari 2019, vervolg:

b. (6) Schrijf een C++-functie `verwijder (ingang)` die het eerste object uit de lijst met ingang `ingang` netjes verwijdert, indien dit bestaat.

c. (4) Schrijf een C++-functie `hoogop (ingang)` die als er minstens twee objecten zijn en als het `info`-veld van het eerste object oneven is, dit ophoogt met het `info`-veld van het tweede object. In het voorbeeld: 7 wordt 42.

d. (3) In de functies bij **a**, **b** en **c** staat in de heading een pointer. Deze heb je call by value of call by reference doorgegeven (met een `&`). Maakt het voor de werking van deze functies verschil uit of die `&` erbij staat? Mag het, moet het? Leg duidelijk uit.

e. (6) Schrijf een C++-functie `repareer (ingang)` die ervoor zorgt dat na afloop alle `volg1`-pointers naar het volgende, en alle `volg2`-pointers naar het vorige object wijzen.

www.liacs.leidenuniv.nl/~kosterswa/pm/tentamens.php

Werkcollege Programmeermethoden 16/17 december 2021

Tentamen 5 januari 2016

1. In een array `int A[n]` staan `n` (een `const` ≥ 1) gehele getallen > 0.
a. (5) Schrijf een Booleaanse C++-functie `hoe (A,X,n)` die bepaalt of er een getal dat *hooguit* 1 van het gehele getal `X` verschilt in het array `A` voorkomt. Gebruik geen `(f)abs`.
b. (8) Schrijf een C++-functie `int langste (A,n,gem)` die de lengte uitrekent van een langste stijgende (of beter: niet-dalende) aaneengesloten deelrij in het array `A`. Hierbij moet `gem` het op de gebruikelijke manier naar een geheel getal afgeronde gemiddelde zijn van de getallen in de betreffende deelrij. Voor het array `1 7 1 1 2 7 6 3` zou het antwoord 4 zijn, namelijk de lengte van de deelrij `1 1 2 7`. En `gem` moet 3 worden (via 11/4). Als er meer deelrijen dezelfde maximale lengte realiseren: het gemiddelde van de eerste.
c. (8) Schrijf een C++-functie `busort (A,n)` die het array `A` met de volgende variant van *bubblesort* oplopend sorteert. In de eerste ronde wordt het gehele array van links naar rechts doorlopen, in de tweede ronde van rechts naar links, in de derde van links naar rechts, etcetera. Stop als er tijdens een ronde geen verwisselingen waren.
d. (4) Hoeveel vergelijkingen tussen array-elementen worden minimaal/maximaal uitgevoerd in de functie van **c**? Geef voor beide situaties een voorbeeld.

2. (25) **a.** (6) Bij een functie kan je te maken hebben met *call by value* en *call by reference*, en ook met *locale* en *globale* variabelen. Leg deze vier verschillende en ook nog *formele* en *actuele* parameters. Leg deze zes begrippen duidelijk uit.
b. (6) Gegeven een C++-programma met daarin de volgende twee functies:

```
int hillary (int n, int m) {
  n--; return n+m-1; }//hillary
int donald (int n, int m) {
  int a = 6; m--; n += 2; a++;
  m = n + hillary (n,m) + hillary (m,n) + a;
  cout << a << "," << m << "," << n << endl; return a+n-m; }//donald
```

Verder zijn de globale variabelen `a` en `m` gegeven, beide van type `int`. Wat is dan de uitvoer van het volgende stukje programma (leg je antwoord duidelijk uit):

```
a = 1; m = 4; cout << donald (m,a) << endl;
cout << a << "," << m << endl;
```

c. (5) We voegen nu vier maal een `&` toe bij de parameters in de heading van `hillary` en `donald`. Beantwoord opnieuw vraag **b**; leg uit waarom verschillende uitkomsten mogelijk zijn, en geef deze.
d. (4) Als in de functie `hillary` ergens `a = donald (42,a-a);` staat, compileert het programma dan nog? Onderscheid gevallen met en zonder `&`.
e. (4) Geef een eenvoudige uitdrukking voor de functiewaarde van `donald (r,s)`, uitgedrukt in `r` en `s`, voor de situatie zonder `&`'s (net zoals bij **b**).

3. (25) Gegeven is een `m` bij `n` (beide `const` > 0) array `M` met gehele getallen ≥ 0. Hierbij geeft `M[i][j]` ≥ 0 het aantal mensen ter plaatse `(i,j)` aan (met $0 \leq$ `i` $<$ `m` en $0 \leq$ `j` $<$ `n`). Zie hiernaast voor een voorbeeld met `m` $= 4$ en `n` $= 5$. De constantes `m` en `n` hoeven bij deze opgave niet doorgegeven te worden als parameter.

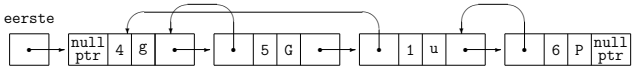
3	0	4	2	0
7	2	4	4	2
1	2	0	4	1
6	9	3	1	0

a. (7) Schrijf een C++-functie `int druk (M,som)` die de index van de drukste kolom, dat wil zeggen de kolom met de grootste som, geeft. In `som` moet de desbetreffende som komen. In het voorbeeld kolom 0, met `som` gelijk aan 17. Als er meerdere kolommen dit maximum realiseren, geef degene met de hoogste kolomindex.
b. (8) Schrijf een C++-functie `int duos (M)` die het aantal horizontaal of verticaal direct aan elkaar grenzende (eventueel overlappende) paren geeft, waarbij de een precies het dubbele van de ander is. In het voorbeeld: 4–2, 2–4, 4–2, 1–2, 2–4, 2–1: 6 stuks.
c. (10) We lopen als volgt door het array `M`. Start bij `(i,j)`. De waarde van dit array-element geeft aan hoeveel plaatsen naar rechts je gaat; de waarde van de nieuwe plek bepaalt hoeveel plaatsen je omlaag gaat, enzovoorts. Als je rechts het array uitloopt kom je in dezelfde rij links weer binnen, en analoog voor de verticale richting. De wandeling stopt als je ergens komt waar je al eerder bent geweest (in het bijzonder bij een verplaatsing van 0). Schrijf een C++-functie `int aantal (M,i,j)` die het aantal stappen uitrekent; in `i` en `j` moeten de coördinaten van het laatst bezochte punt komen. In het voorbeeld, beginnend bij $(0,0)$ eerst 3 naar rechts, dan 2 omlaag, dan 4 naar rechts, dan 0 omlaag en klaar: antwoord 4, en $(\texttt{i},\texttt{j}) = (2,2)$. Hint: gebruik een Booleaans hulparray.

4. (25) Gegeven is het volgende type:

```
class kamer { public: kamer* vorig; int nr; char naam; kamer* volg; };
```

Hierin wordt een lijst van kamers gemaakt (`naam` is een kleine letter of hoofdletter). Het veld `volg` bevat een pointer naar het volgende kamer-object en `vorig` naar het vorige *of* voor-vorige, *of* `nullptr` als die er beide niet zijn. Een voorbeeld (`eerste` van type `kamer*`):



a. (5) Schrijf een C++-functie `voegtoe (eerste,kamernr,kamernm)` die een nieuw `kamer`-object met `kamernr` en `kamernm` erin vooraan in de lijst met ingang `eerste` toevoegt. Zet de `vorig`-pointer van het oude voorste object (als dat bestond) ook goed.
b. (5) Schrijf een C++-functie `verwijder (eerste)` die het voorste `kamer`-object uit de structuur die door `eerste` wordt aangewezen, netjes verwijdert — mits het bestaat. Zet eventuele `vorig`-pointers die er naar wijzen goed.
c. (5) Schrijf een C++-functie `wissel (eerste)` die de kamer-nummers van de twee voorste kamers omwisselt, mits de ene letter de hoofdletter van de andere is (zoals in het voorbeeld; 4 en 5 worden verwisseld). Controleer of de lijst minstens twee objecten heeft.
d. (4) In de functies bij **a**, **b** en **c** staat in de heading een pointer. Deze heb je call by value of call by reference doorgegeven (met een `&`). Maakt het voor de werking van deze functies verschil uit of die `&` erbij staat? Mag het, moet het? Leg duidelijk uit.
e. (6) Schrijf een C++-functie `herstel (eerste)` die alle `vorig`-pointers, behalve die van de voorste twee objecten (zo die al bestaan), naar het voor-vorige object laat wijzen. In het voorbeeld zou de pointer bij het object met `P` erin naar het object met `G` erin moeten gaan wijzen, verder verandert er niets.

Zie www.liacs.leidenuniv.nl/~kosterswa/pm/tentamens.php voor meer vragen en antwoorden!

Opgave 2 van het tentamen van 4 januari 2013:

b. Gegeven een C++-programma met daarin de volgende twee functies:

```
bool mark (int a, int b) {
  int z;
  a = a + b; b = a - b; a = a - b; cout << "M" << a << "," << b << endl;
  z += 10; return ( a < b );
}//mark
int diederik (int b, int a) {
  bool temp; if ( mark (b,a) ) a += 2;
  while ( a > 0 ) { temp = mark (a,b); a--; cout << a << "," << b << endl; }
  z += 10; return ( a + b + 2 );
}//diederik
```

Verder zijn de globale variabelen `x`, `y` en `z` gegeven (van type `int`). Wat is dan de uitvoer van het volgende stukje programma (leg je antwoord duidelijk uit; tip: 9 komma's):

```
x = 1; y = 3; z = 1; x = diederik (y,z); cout << x << "," << y << "," << z;
```

c. Als **b**, maar nu met een `&` ("ampersand") bij de vier parameters van de functies.

d. Als **c**, dus met vier `&`'s erbij, voor:

```
x = 1; y = 3; z = 1; y = diederik (y,y); cout << x << "," << y << "," << z;
```

e. Als in `mark` ergens `a = diederik (static_cast<int>(mark (a,b)),y);` staat, compileert het programma dan nog? Onderscheid gevallen met en zonder `&`.

Uitwerking Opgave 2b

tijd ↓

"main"			"diederik"			"mark"		
x	y	z	a	b	temp	a	b	
1	3	1						
			1	3	?			aanroep `diederik`
						3	1	eerste aanroep `mark` (`z` doet er niet toe)
						1	3	cout: M1,3; geeft `true` terug
			3					"`a` += 2;" uit `diederik`
						3	3	tweede aanroep `mark`
						3	3	cout: M3,3; geeft `false` terug
			2		`false`			cout: 2,3
						2	3	derde aanroep `mark`
						3	2	cout: M3,2; geeft `false` terug
			1		`false`			cout: 1,3
						1	3	vierde aanroep `mark`
						3	1	cout: M3,1; geeft `false` terug
			0		`false`			cout: 0,3
		11						geeft 5 terug aan `x`
5								cout: 5,3,11

"main"			"mark"	tijd ↓
x	y = b uit diederik	z = a uit diederik	temp	
1	3	1		
			?	aanroep diederik
	= a uit mark	= b uit mark		eerste aanroep mark
	1	3		cout: M1,3; geeft true terug
		5		"a += 2;" uit diederik
	= b uit mark	= a uit mark		tweede aanroep mark
	5	1		cout: M1,5; geeft true terug
		0	true	cout: 0,5
		10		geeft 17 terug aan x
17				cout: 17,5,10

"main"			"mark"	
x	y	z	temp	tijd ↓
	= a uit diederik			
	= b uit diederik			
1	3	1		
			?	aanroep diederik
	= a uit mark			aanroep mark
	= b uit mark			
	0			cout: M0,0; geeft false terug
		11		geeft 2 terug aan y
	2			cout: 1,2,11

e. Dit mag alleen als er geen & staat voor de eerste parameter van diederik, en er boven mark een prototype van diederik wordt gezet.

- werk wellicht nog aan de vierde programmeeropgave — de deadline is al voorbij!

- werkcollege 16/17 december 2021: oude tentamens

- tentamen:
  woensdag 5 januari 2022, 14:15–17:15 uur, Gorlaeus zalen 2 (extra tijd) en 4/5
  hertentamen:
  donderdag 31 maart 2022, 14:15–17:15 uur, Gorlaeus zalen 1 en 2

- www.liacs.leidenuniv.nl/~kosterswa/pm/