Programmeermethoden

Functies & files

Walter Kosters en Jonathan Vis

week 4: 23–27 september 2024

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

ma	di	wo	donderdag 26.9	vrijdag
			11:00–12:45 Gorlaeus zaal 3 college 4 iedereen	13:15–. . . Gorlaeus BW.0.17/18/19 werkcollege 4 (*), vragenuur Wiskundigen
			13:15–. . . Snellius DM.0.09/13/21 werkcollege 4 (*), vragenuur Informatici	

(*) beter bekend als het functie-practicum

Vragenuren maandag 30 september vanaf 15:00 uur en woensdag 2 oktober vanaf 13:00 uur bij BM.2.07; analoog de week daarna. (Op 3 en 4 oktober geen (werk)colleges.)

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;
```

Een eenvoudige `void`-functie:

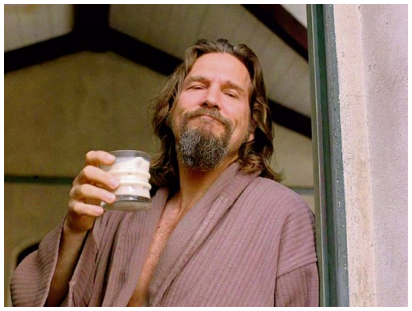
```
// kort infoblokje
void infoblokje ( ) {
  cout << "Hallo allemaal ..." << endl;
  cout << "Enzovoorts ..." << endl;
}//infoblokje

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

Let op het inspringen.

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
```

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, 2023 niet, 2024 wel, 2000 wel, en 2100 niet . . .

De grootste gemeenschappelijke/gemene deler (ggd) van twee positieve gehele getallen (≥ 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.

Functies hebben verschillende soorten parameters:

- globaal — gelden overal — te vermijden!
- 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

En met een globale variabele erbij:

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

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 Lychrel-vermoeden enigszins controleert voor 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.

Tekstfile kopiëren

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 staat hier het UNIX-commando `cp`.)

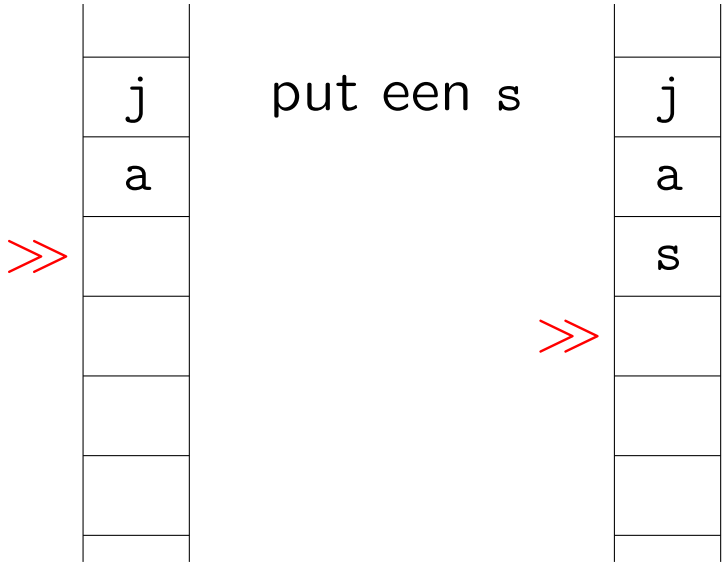
Wijzig stap voor stap zo'n kopieerprogramma:

```
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) ...
```

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␣␣>␣12560/196␣)␣␣{{␣//␣commentaar!
cout␣<<␣"Oud";//␣>64␣of␣niet?
```

moet worden:

```
if␣(␣leeftijd␣␣>␣12560/196␣)␣␣{{␣
␣␣␣␣␣␣cout␣<<␣"Oud";
```

Hier is ␣ een spatie. En "tab = 3".

Is $196 \to 196 + 691 = 887 \to \ldots$ een Lychrel-getal?
Hoe vaak komt `mag` voor?

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

Tweede programmeeropgave

Programmeermethoden

Home Algemeen Materiaal Opdrachten en Cijfers

Opdrachten en Cijfers

Programmeermethoden 2024
Tweede programmeeropgave: Netjes

De *tweede* programmeeropgave van het vak Programmeermethoden in het najaar van 2024 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 (0 als `d` < 0). 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.
   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. Schrijf ook zelf een functie die een hoofdletter naar de bijbehorende kleine letter converteert.
4. Hebben we misschien met Lychrel-getallen te maken?
   Voor elk geheel getal > 0 uit de invoerfile wordt gekeken of het een Lychrel-getal is. Je telt herhaald het getal bij diens omgekeerde op, bijvoorbeeld 196 plus 691 wordt 887, dan 887 plus 788 wordt 1775, enzovoorts. Je stopt als je een palindroom hebt (dat is een getal dat gelijk is aan diens omgekeerde, bijvoorbeeld 5372735, of 200002, of 7); dan is het oorspronkelijke getal *geen* Lychrel-getal. Je stopt ook als je boven `INT_MAX` uitkomt, en dan weet je het niet. Op het scherm wordt het oorspronkelijke getal afgedrukt, en wat het aantal iteraties is om bij een palindroom uit te komen (bijvoorbeeld 0 voor 545, en 1 voor 113), of het nummer van de iteratie waarvan het resultaat boven `INT_MAX` (gebruik `include <climits>`) uitkomt. Als dit laatste gebeurt (bijvoorbeeld bij 196 tijdens de 18de stap), 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. Neem aan dat de getallen in de invoerfile tussen 1 en `INT_MAX`, grenzen inbegrepen, zitten.

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. 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!
Twee voorbeeldfiles:

1. File simpel2024.txt moet worden simpel2024uit.txt, als `tab` gelijk aan 3 is. Accolades niet goed gepaard.
2. File lastig2024.cc moet worden lastig2024uit.cc, als `tab` gelijk aan 3 is. Het drieletterwoord `the` komt 6 keer voor, en het 3 keer. Accolades goed gepaard.

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 een getal, Lychrel-test, en behandelen 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 14 oktober 2024, 18:00 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 via Brightspace > Course Tools > Assignments. Stuur geen executable's, LaTeX-files of PDF-files, lever alleen één C++-file digitaal in!
- Doe een print van het verslag in de doos bij kamer Gorlaeus BM.2.07.

De laatst voor de deadline ingeleverde versie wordt nagekeken.
Overal duidelijk datum en namen van de (maximaal twee) makers vermelden, in het bijzonder als commentaar

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

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

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

← functies

files →

```
// file verwerken, hier kopieren
ifstream invoer ("invoer.txt");
ofstream uitvoer ("uitvoer.txt");

char kar = invoer.get ( );
while ( ! invoer.eof ( ) ) {
  uitvoer.put (kar);
  // ...
  kar = invoer.get ( );
}//while

invoer.close ( );
uitvoer.close ( );
```

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

video

```
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
```

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
```

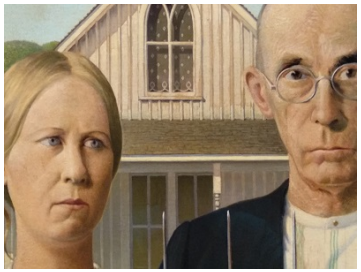
Neem aan dat `getal` minstens 1 is.

Een getal omkeren ($196 \rightarrow 691$) of het aantal cijfers van een getal tellen ($6171 \rightarrow 4$) gaat analoog!

Geen strings, geen arrays!

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

```
␣␣if␣(␣leeftijd␣␣>␣12560/196␣)␣␣{{␣//␣commentaar!
cout␣<<␣"Oud";//␣>64␣of␣niet?
```



moet worden:

```
if␣(␣leeftijd␣␣>␣12560/196␣)␣␣{{␣
␣␣␣␣␣␣cout␣<<␣"Oud";
```

Hier is ␣ een spatie. En "tab = 3".

Is $196 \to 196 + 691 = 887 \to \ldots$ een Lychrel-getal?
Hoe vaak komt `mag` voor?

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. daarna, of juist eerder, de Lychrel-getallen: `INT_MAX`!

4. en tot slot details (drieletterwoorden), tellers, . . . , en het verslag

↙ $\leq \approx$ 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

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` staat/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 14 oktober 2024, 18:00 uur

video

- op 3/4 oktober geen (werk)college: Leidens ontzet

vragenmiddagen
maandag 30.9,
15:00–. . .
woensdag 2.10,
13:00–. . .
bij BM.2.07

- 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 opgavendictaat
- doe nu het `vierde werkcollege`: het functie-practicum
- `www.liacs.leidenuniv.nl/~kosterswa/pm/`