Programmeermethoden

Types

Walter Kosters en Jonathan Vis

week 2: 9–13 september 2024

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

E-mailadres: pm@liacs.leidenuniv.nl

maandag	di	wo	donderdag 12.9	vrijdag
middag Gorlaeus bij BM.2.07 vragenuur			11:00–12:45 Gorlaeus zaal 3 college 2 iedereen	13:15–. . . Gorlaeus BW.0.17/18/19 werkcollege 2, vragenuur Wiskundigen
			13:15–. . . Gorlaeus DM.0.09/13/21 werkcollege 2, vragenuur Informatici	

Werkcollege Wiskundigen van vrijdag 20 september $\Rightarrow$ donderdag 19 september en/of maandag 23 september.

Eerste programmeeropgave

Programmeermethoden

Home Algemeen Materiaal Opdrachten en Cijfers

Opdrachten en Cijfers

Programmeermethoden 2024
Eerste programmeeropgave: Breuken

De *eerste* programmeeropgave van het vak Programmeermethoden in het najaar van 2024 heet *Breuken*; zie ook het eerste, tweede en derde werkcollege.

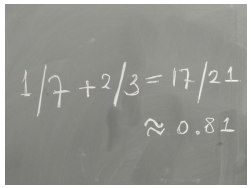
Deze opgave probeert te bepalen of iemand geschikt is voor een studie aan de universiteit. Daartoe moeten enkele vragen beantwoord worden, zo moet je als kandidaat weten op welke dag je geboren bent. En als je niets van breuken weet, is een beta-studie misschien niet verstandig.

Om te beginnen moet de gebruiker diens geboortejaar als geheel getal invoeren, en daarna de geboortemaand, ook als geheel getal. Vervolgens voert men de geboortedag in, wederom als geheel getal. Het programma berekent dan de leeftijd, 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 *23 september 2024* 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 diens 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 minstens twee maal "netter" aangesproken dan jongeren. Splits de C++-code in het programma niet onnodig vaak!

Er wordt gekeken of de aanstaande student breuken bij elkaar kan optellen of van elkaar kan aftrekken. 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.

De gebruiker moet twee willekeurige gegeven breuken, met tellers en noemers tussen 1 en 19, bij elkaar optellen of van elkaar aftrekken (dat laatste staat erbij vermeld). De gebruiker moet het antwoord twee maal geven: als twee gehele getallen (het eerste getal kan negatief zijn; de breuk die hierdoor wordt voorgesteld moet in principe goed zijn; er mag hierbij niet vereenvoudigd worden, uitgaande van het "gewone" optel-algoritme) en als decimaal getal, met een decimale punt, dat er maximaal een constante MAXFOUT (zeg 0.1) naast mag zitten. Als de originele getallen bijvoorbeeld 1 / 7 en 2 / 3 zijn, is het eerste antwoord 17 en 21 (voorstellende de breuk 17 / 21), en wordt bij het tweede 0.81 goed gerekend. Nog een voorbeeld: 2 / 2 min 6 / 3 wordt -6 / 6. Converteer zelf op de juiste manier tussen int en double. Liefhebbers mogen de resulterende breuk vereenvoudigen (zie het vierde college).



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. Wederom, 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: men voert niet al te gekke getallen of letters in, etcetera. Vanzelfsprekend worden 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 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 23 september 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!i
- 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 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.
Vragen: e-mail naar `pm@liacs.leidenuniv.nl`

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

richtlijnen

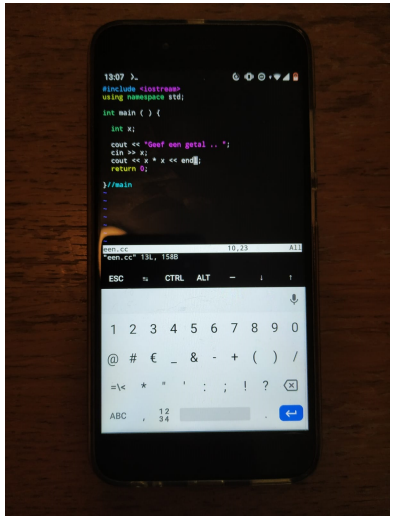
Voor we aan de types beginnen, eerst nog een "uitgebreide" stoomcursus UNIX en wat meer over C++.

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)
`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
(`lpr`	file printen)

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. Verdere controle: `ps`, `top` en `kill`. Voorbeeld, met pipelining: `ps -eaf | grep ikkuh`. (En CTRL-Z is iets anders.)

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 "; // want twee statements hier
    cout << PIE * straal * straal << endl;
  }//if
  else { // hier accolades niet nodig, maar liever wel!
    cout << "Niet zo negatief ..." << endl;
  }//else
  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.

We scheiden berekenen en printen = afdrukken.
En variabelen graag "bovenin" aanmaken.

Elk C++-programma bestaat uit:

- speciale symbolen als `+`, `-`, `*`, `/`, `%`, `>=`, `<=`, `<<`, `>>`, `=` (“wordt”), `==` (“is gelijk aan”), `!=` (“is ongelijk 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
```

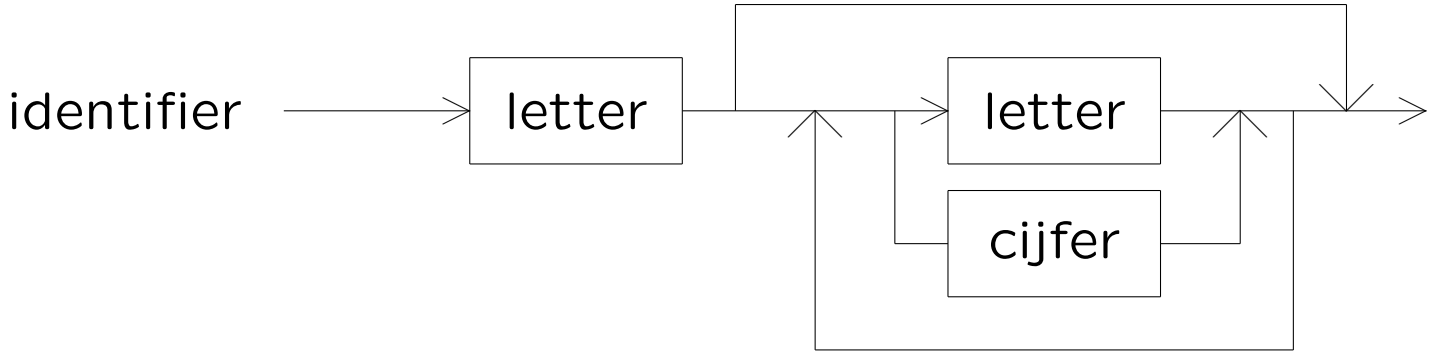
En uit de ANSI-ISO C++-standaard:

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

Er zijn er nog minstens 30 meer. Huidige versie: C++23.

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.

Zeker 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++, 12.9.2024, g++ versie 3.14159
// ...
...
...
...
  cout << "Welkom ... " << endl; // infoblokje
  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, op 1 regel!?
  if ( leeftijd < 18 ) {
    school ( );
  }//if
  else {
    if ( leeftijd > 67 ) {
      pensioen ( );
      return 1; // stopt het programma
    }//if
    werken ( );
  }//else
  return 0;
}//main
```

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

```
int main ( ) {
12int leeftijd; cin >> leeftijd; // foei, op 1 regel!?
12if ( leeftijd < 18 ) {
1234school ( );
12}//if
12else {
1234if ( leeftijd > 67 ) {
123456pensioen ( );
123456return 1; // stopt het programma
1234}//if
1234werken ( );
12}//else
12return 0;
}//main
```

richtlijnen

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)
```

Reële getallen — 1

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 = "rechts" afronden), `floor` (idem, naar beneden = "links") 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 = 0.932;  // 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 $\leftarrow$) 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 ) ...  // FOUT, nu wordt p eerst true,
   // is de test altijd waar, en is p ook nog veranderd
if ( getal == 2 || 3 || 5 || 7 || 11 || 13 ) ... // FOUT
   // 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, deze week zou minstens het datumgedeelte af moeten zijn — de deadline is maandag 23 september 2024, 18:00 uur!
- verslag (in LaTeX) $\longrightarrow$ volgende week
- lees Savitch Hoofdstuk 1 nog eens, begin dan 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/