
Programmeermethoden

Types

Walter Kusters

week 2: 12–16 september 2011

<http://www.liacs.nl/home/kusters/pm/>

Voor we aan de **types** beginnen, eerst nog

- een “uitgebreide” stoomcursus UNIX
- wat meer over C++

En op de Mac:

`http://www.liacs.nl/home/kosters/pm/macsjoerd.php`

De belangrijkste commando's zijn:

<code>ls</code>	overzicht files in directory
<code>lpr</code>	file printen
<code>more</code>	file op beeldscherm
<code>rm</code>	file verwijderen
<code>cp</code>	file kopiëren
<code>mv</code>	file verplaatsen
<code>cd</code>	van directory veranderen
<code>chmod</code>	rechten bij files regelen
<code>man</code>	hulp

Enkele voorbeelden:

```
ls -lrt; man ls; chmod 644 *; cp een twee  
En in Wiskunde-zalen: lpr -Php307 file.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
```

Elk C++-programma bestaat uit:

- speciale symbolen als `+`, `%`, `>=`, `=` (**wordt**), `==` (**is**)
- woordsymbolen als `if`, `while`
- identificers als `int`, `straal`; vaak zelfgemaakt
- getallen als `42`, `3.14159`
- 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 **keywords** van C++:

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

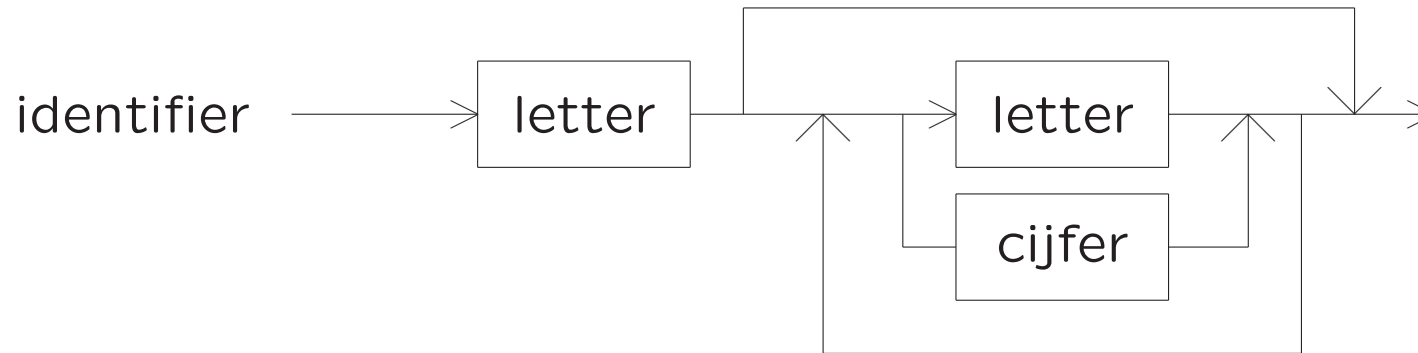
En uit de ANSI-ISO C++-standaard:

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

Er zijn er nog circa 30 meer.

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

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



Zie verder: Noam Chomsky, BNF (Backus Naur Form), kontekstvrije/formele talen — “loodspet”.

Een goed programma bevat veeeeeeeeeeeeel **commentaar**:

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

of (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)!

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
```

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.

C++ kent de volgende basistypes (**data-types**):

- `int` — geheel getal
- `double` (of `float`) — *benadering* van reëel getal
- `bool` — waar (`true`) of niet-waar (`false`)
- `char` — karakter (lettertje)

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)

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>`.

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

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

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 \mathbb{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 $\lfloor -6.8 \rfloor = -7$.

```
double x; // een reeel getal
double temp = 36.5, dollar = 0.77513; // 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 ...

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 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 karacters gebruiken een **escape sequence**: `'\n'`.
Ook nog: `'\t'` (tab), `'\\'` (backslash), ...
Zo geeft `cout << "\\n";` op het beeldscherm `\n`.

Een nieuwe regel op het beeldscherm kun je maken met `cout << "\\n";` of (beter) met `cout << endl;`. Dit laatste “flusht” ook nog — en dat is soms fijn.

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

```
int getal; bool p;
getal = 17;
p = false;
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 ( '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 als $x = y = 5;$, en dat betekent $x = (y = 5);$. Nu worden x en y beide 5: 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 vrijdag 23 september 2011 **vragen??**
- lees Savitch Hoofdstuk 1 (nog eens), begin met 2
- lees dictaat Hoofdstuk 1, 2 (nog eens), begin met 3
- maak opgaven 1/5 uit het “opgavendictaat”
- vragenuren: di/wo/do, 15.30–17.00, computerzalen
- <http://www.liacs.nl/home/kosters/pm/>