

Gertjan Laan

AAN DE SLAG MET C++

Boom

Zevende druk



Aan de slag met C++

Gertjan Laan

Boom

**inclusief
website!**

Met behulp van onderstaande unieke activeringscode krijg je toegang tot de website www.aandeslagmetcpp.nl voor extra materiaal. Deze code is persoonsgebonden en gekoppeld aan de 7^e druk. Na activering van de code is de website twee jaar toegankelijk. De code kan tot zes maanden na het verschijnen van een volgende druk geactiveerd worden. De code is eenmalig te gebruiken.

Opmaak binnenwerk: Holland Graphics, Amsterdam
Basisontwerp omslag: Dog & Pony, Amsterdam
Omslagontwerp: Coco Bookmedia, Amersfoort
Beeld omslag: Olga Salt/Shutterstock
Beeld hoofdstukopener: Nickolay Grigoriev/Shutterstock

© Gertjan Laan & Boom uitgevers Amsterdam, 2021

Behoudens de in of krachtens de Auteurswet gestelde uitzonderingen mag niets uit deze uitgave worden verveelvoudigd, opgeslagen in een geautomatiseerd gegevensbestand, of openbaar gemaakt, in enige vorm of op enige wijze, hetzij elektronisch, mechanisch, door fotokopieën, opnamen of enige andere manier, zonder voorafgaande schriftelijke toestemming van de uitgever.

Voor het overnemen van (een) gedeelte(n) uit deze uitgave in bijvoorbeeld een (digitale) leeromgeving of een reader in het onderwijs (op grond van artikel 16, Auteurswet 1912) kan men zich wenden tot Stichting Uitgeversorganisatie voor Onderwijslicenties, Postbus 3060, 2130 KB Hoofddorp, www.stichting-uvo.nl.

No part of this book may be reproduced in any form, by print, photoprint, microfilm or any other means without written permission from the publisher.

ISBN 9789024438617
ISBN e-book 9789024438624
NUR 173

www.aandeslagmetcpp.nl
www.boomhogeronderwijs.nl

Inhoudsopgave

	Voorwoord bij de zevende druk	19
1	Introductie	21
1.1	Inleiding	21
1.2	De preprocessor, compiler en linker	21
1.3	Eerste voorbeeld	23
1.3.1	<i>Toelichting bij het eerste voorbeeld</i>	24
1.3.2	<i>Declaratie van variabelen</i>	25
1.4	Namespaces	27
1.4.1	<i>Over using namespace std</i>	29
1.5	Getaltypen	29
1.5.1	<i>Expressies: de operatoren +, -, *, / en %</i>	32
1.5.2	<i>Typeconversie</i>	33
1.5.3	<i>De typecast</i>	34
1.5.4	<i>Assignment-operatoren</i>	35
1.5.5	<i>Uniforme initialisatie</i>	36
1.5.6	<i>De increment-operator ++ en de decrement-operator --</i>	37
1.5.7	<i>De naam C++</i>	38
1.5.8	<i>Postfix en prefix</i>	38
1.5.9	<i>Prioriteiten</i>	39
1.6	Literals en constanten	39
1.6.1	<i>Constanten</i>	41
1.6.2	<i>Constante expressie: constexpr</i>	42
1.7	De scope van variabelen en constanten	43
1.8	Het type char	44
1.8.1	<i>ASCII-code</i>	45
1.8.2	<i>Escape sequences</i>	46
1.8.3	<i>Literals van het type char</i>	48
1.9	De typeaanduiding auto en decltype	49
1.10	De opmaak van de broncode	50
1.11	Samenvatting	50
1.12	Vragen	51
1.13	Opgaven	53
2	Selecties en herhalingen	57
2.1	Inleiding	57

2.1.1	<i>Relationele operatoren</i>	57
2.1.2	<i>Logische operatoren: de en-operator &&</i>	58
2.1.3	<i>Een bool-variabele</i>	60
2.2	Het if-statement	61
2.3	Het if-else-statement	63
2.3.1	<i>Over cin</i>	65
2.4	Het switch-statement	65
2.4.1	<i>Wat niet kan met een switch-statement</i>	67
2.5	Het for-statement	68
2.5.1	<i>Controlegedeelte van het for-statement</i>	69
2.5.2	<i>Zet geen puntkomma na het controlegedeelte</i>	71
2.5.3	<i>De scope van de controlevariabele</i>	71
2.6	Recht onder elkaar zetten van gehele getallen	72
2.6.1	<i>De manipulator setfill()</i>	74
2.6.2	<i>De manipulatoren hex, oct en dec</i>	75
2.6.3	<i>Het gebruik van setw() met tekst</i>	76
2.6.4	<i>Links of rechts uitlijnen</i>	77
2.6.5	<i>Het uitvoerformaat van floating-pointgetallen</i>	78
2.7	Variaties met een for-statement	79
2.7.1	<i>For-statement waarvan de body niet wordt uitgevoerd</i>	82
2.8	Het enumerated type	83
2.8.1	<i>strongly typed (scoped) enumerations</i>	85
2.9	Het while-statement	86
2.9.1	<i>Syntax van het while-statement</i>	86
2.10	Het do-while-statement	88
2.10.1	<i>De invoerbuffer</i>	90
2.11	De oneindige loop	90
2.12	Het break-statement	92
2.13	Controleren van de invoer	94
2.14	Het continue-statement	95
2.15	Algoritmen: een loop binnen een loop	97
2.15.1	<i>Genest for-statement</i>	97
2.15.2	<i>Het omgekeerde probleem</i>	98
2.16	Samenvatting	102
2.17	Vragen	102
2.18	Opgaven	104
3	Funcities	111
3.1	Inleiding	111
3.2	Een eenvoudige functie	111
3.2.1	<i>Prototype</i>	112
3.2.2	<i>Implementatie van de functie</i>	112
3.2.3	<i>Funcieaanroep</i>	113
3.3	Een functie met argumenten	113
3.3.1	<i>Voordeel van een functie met argumenten</i>	116

3.3.2	<i>Actuele en formele argumenten</i>	116
3.3.3	<i>Nog een functie</i>	117
3.3.4	<i>Defaultargumenten</i>	118
3.4	Wiskundige functies	119
3.4.1	<i>C++-library</i>	119
3.5	Functies die een waarde afleveren	121
3.6	Gestructureerd programmeren en functies	122
3.7	Functies die geen waarde afleveren	125
3.8	Een constexpr-functie	125
3.9	Richtlijnen bij het schrijven van functies	126
3.10	Prototyping, aanroep, implementatie en volgorde	126
3.10.1	<i>Het prototype of de declaratie van de functie</i>	127
3.10.2	<i>De functieaanroep</i>	127
3.10.3	<i>De definitie of implementatie van de functie</i>	128
3.10.4	<i>De volgorde van functies</i>	128
3.10.5	<i>Splitsen van header en implementatie</i>	129
3.11	Lokale variabelen	130
3.11.1	<i>De argumenten van een functie</i>	132
3.12	Statische en globale variabelen	132
3.12.1	<i>Statische variabelen</i>	132
3.12.2	<i>Initialisatie van statische variabelen</i>	134
3.12.3	<i>Globale variabelen</i>	134
3.12.4	<i>Globale en lokale variabelen met dezelfde naam</i>	136
3.13	Reference-argument	136
3.13.1	<i>Wanneer gebruik je een reference-argument?</i>	138
3.14	Uitbreiding van de richtlijnen voor het schrijven van een functie	139
3.14.1	<i>De functie verwissel()</i>	140
3.15	Een functie die een referentie aflevert	142
3.16	Functieoverlading	145
3.16.1	<i>Overladen op grond van aantal argumenten</i>	145
3.16.2	<i>Pre- en postcondities</i>	147
3.16.3	<i>Overladen op grond van type van de argumenten</i>	147
3.17	Samenvatting	148
3.18	Vragen	149
3.19	Opgaven	150
4	C-arrays en pointers	153
4.1	Inleiding	153
4.2	Het geheugen	153
4.2.1	<i>Bytes</i>	153
4.2.2	<i>Bits</i>	154
4.2.3	<i>Adressen</i>	155
4.3	Arrays	157
4.3.1	<i>Een C-array van doubles</i>	159

4.3.2	<i>De bovengrens van een array</i>	160
4.3.3	<i>Initialisatie van een C-array</i>	161
4.3.4	<i>De grootte van een C-array bepalen met sizeof</i>	162
4.3.5	<i>C-arrays en andere variabelen tegelijk declareren</i>	163
4.4	Algoritme: zoeken van de grootste waarde in een C-array	163
4.5	C-array als argument van een functie	164
4.5.1	<i>Nogmaals: zoeken van de grootste waarde in C-array</i>	165
4.6	Een const-arrayargument	166
4.6.1	<i>Algoritme: zoeken van een getal in een C-array</i>	167
4.7	Range-based for	168
4.8	Tweevoudige arrays	169
4.8.1	<i>Initialiseren van een tweevoudige C-array</i>	170
4.8.2	<i>Een tweevoudige C-array als argument van een functie</i>	172
4.8.3	<i>Een speelbord voor boter, kaas en eieren</i>	174
4.9	Voor- en nadelen van C-arrays	177
4.10	Pointers	179
4.10.1	<i>Opmerkingen over notaties bij het gebruik van pointers</i>	181
4.11	De elementen van een C-array langslopen met een pointer	183
4.11.1	<i>Opschuiven van een pointer naar double</i>	185
4.11.2	<i>Rekenen met pointers</i>	186
4.11.3	<i>De betekenis van *p++</i>	186
4.12	Pointers en arraynotatie	187
4.13	Een nullptr	188
4.14	Een adres als functiewaarde	189
4.14.1	<i>Pointers naar een constante</i>	191
4.14.2	<i>Pointer-constante</i>	192
4.15	Een const-pointer als argument	193
4.16	Pointer naar een functie	195
4.17	Typedef	198
4.17.1	<i>Typedef voor functiepointer</i>	199
4.18	Samenvatting	200
4.19	Vragen	201
4.20	Opgaven	201
5	Strings en vectoren	207
5.1	Inleiding	207
5.2	C-strings	207
5.3	Een string-object	208
5.3.1	<i>Invoer van strings vanaf het toetsenbord</i>	210
5.3.2	<i>Combineren van cin en getline()</i>	211
5.3.3	<i>Invoerbuffer leegmaken</i>	213
5.4	Samenvoegen en conversie	214
5.4.1	<i>Uitvoer naar een string met een stream</i>	215
5.5	Een paar functies van de klasse string	216
5.5.1	<i>De lidfuncties length() en size()</i>	216

5.5.2	<i>De lidfunctie substr()</i>	216
5.5.3	<i>De lidfunctie replace()</i>	217
5.5.4	<i>De lidfunctie find()</i>	217
5.5.5	<i>De lidfunctie c_str()</i>	219
5.6	Vergelijken van strings	219
5.7	Een iterator voor een string	220
5.7.1	<i>Range-based for-statement en string</i>	222
5.7.2	<i>Omzetten van een string in hoofdletters</i>	222
5.7.3	<i>String omkeren</i>	224
5.8	Vectoren	227
5.8.1	<i>Een vector declareren en vullen</i>	227
5.8.2	<i>Iterator voor een vector</i>	228
5.8.3	<i>Auto, range-based for-statement en vector</i>	229
5.8.4	<i>Een vector initialiseren</i>	230
5.8.5	<i>Kopiëren van een vector</i>	232
5.8.6	<i>Een printfunctie voor een int-vector</i>	232
5.8.7	<i>De indexoperator van een vector</i>	234
5.8.8	<i>Het groeien van een vector</i>	235
5.9	Functies van de klasse <code>std::vector</code>	237
5.10	De containerklasse <code>array</code>	238
5.11	Samenvatting	239
5.12	Vragen	240
5.13	Opgaven	240
6	Klassen maken	243
6.1	Inleiding	243
6.2	Een klasse voor bankrekeningen	243
6.2.1	<i>Klassendiagram</i>	244
6.2.2	<i>De broncode van de klasse Bankrekening</i>	244
6.2.3	<i>Een constructor</i>	245
6.2.4	<i>De constructor toevoegen en aanroepen</i>	246
6.2.5	<i>Bankrekening-objecten maken</i>	246
6.2.6	<i>Uniforme initialisatie</i>	247
6.2.7	<i>De functie to_string()</i>	248
6.2.8	<i>Meer functies voor de klasse Bankrekening</i>	249
6.2.9	<i>De functie stort()</i>	249
6.2.10	<i>De functie neem_op()</i>	250
6.2.11	<i>De functie get_saldo()</i>	250
6.2.12	<i>Het nieuwe klassendiagram van Bankrekening</i>	250
6.2.13	<i>Nieuwe broncode van Bankrekening</i>	251
6.3	Een klasse voor studenten	253
6.3.1	<i>De attributen</i>	253
6.3.2	<i>Getters en setters</i>	254
6.3.3	<i>De broncode van de klasse Student</i>	255
6.4	Data hiding	257

6.5	Een klasse voor datums	258
6.5.1	<i>Implementatie van Datum</i>	259
6.5.2	<i>Const lidfuncties</i>	260
6.5.3	<i>Defaultwaarden voor constructor</i>	261
6.6	Relatie tussen klassen	261
6.6.1	<i>Een initialisatielijst</i>	265
6.6.2	<i>Een const reference-argument</i>	266
6.7	Objecten	266
6.8	De kassa	267
6.8.1	<i>Automatische defaultconstructor</i>	268
6.8.2	<i>Attribuut voor de kassa</i>	268
6.8.3	<i>Zelfgemaakte defaultconstructor</i>	269
6.8.4	<i>Een paar functies voor de kassa</i>	269
6.8.5	<i>Lidfuncties buiten de klasse definiëren</i>	269
6.8.6	<i>De scope-operator ::</i>	271
6.8.7	<i>Over inline functies</i>	272
6.9	Over constructors	273
6.9.1	<i>Constructor-overloading</i>	273
6.9.2	<i>Constructor met defaultargumenten</i>	275
6.9.3	<i>Constructor-overloading en defaultargumenten</i>	275
6.9.4	<i>Delegerende (delegating) constructor</i>	276
6.9.5	<i>Initialiseren van een constante in een klasse</i>	277
6.9.6	<i>Directe initialisatie van attributen</i>	277
6.9.7	<i>Een constexpr-constructor</i>	278
6.9.8	<i>Expliciete defaultconstructor</i>	279
6.9.9	<i>Constructor en delete</i>	281
6.10	Het keyword struct	281
6.11	Samenvatting	282
6.12	Vragen	282
6.13	Opgaven	283
7	Objectgeoriënteerd ontwerpen	287
7.1	Inleiding	287
7.2	Teams van studenten	287
7.2.1	<i>De copy-constructor</i>	290
7.2.2	<i>Twee kopieën of niet?</i>	291
7.3	De klasse Team met een vector	292
7.3.1	<i>Klassendiagram met een collectie</i>	294
7.3.2	<i>Compositie en aggregatie</i>	295
7.3.3	<i>Multipliciteiten in UML</i>	296
7.4	Documentatie maken	297
7.4.1	<i>Het schrijven van tekst voor doxygen</i>	298
7.4.2	<i>Genereren van de documentatie</i>	300
7.5	Een winkel	301
7.5.1	<i>Analyse</i>	302

7.5.2	<i>Een lijst van de zelfstandige naamwoorden</i>	302
7.5.3	<i>De klassen</i>	303
7.5.4	<i>De associaties</i>	303
7.5.5	<i>Navigeerbaarheid</i>	304
7.5.6	<i>Het type van de attributen</i>	304
7.5.7	<i>De functies</i>	304
7.5.8	<i>De broncode van de klasse Artikel</i>	305
7.5.9	<i>De broncode van de klasse Catalogus</i>	306
7.5.10	<i>Nogmaals over navigeerbaarheid</i>	307
7.5.11	<i>De broncode van de klasse Bestelling</i>	307
7.6	<i>Sms-dienst</i>	307
7.6.1	<i>Analyse van de sms-dienst</i>	308
7.6.2	<i>De lidfuncties</i>	309
7.6.3	<i>De broncode van SMS</i>	310
7.6.4	<i>De broncode van Provider</i>	311
7.6.5	<i>De pijloperator</i>	312
7.6.6	<i>De broncode van Mobiel</i>	312
7.6.7	<i>Forward declaratie</i>	313
7.7	<i>Samenvatting</i>	314
7.8	<i>Vragen</i>	315
7.9	<i>Opgaven</i>	315
8	Conversie en operatoren	319
8.1	<i>Inleiding</i>	319
8.2	<i>Constructors en conversie</i>	320
8.2.1	<i>Het woord explicit</i>	320
8.2.2	<i>Initialisatie met een string</i>	321
8.2.3	<i>Conversie na de initialisatie</i>	322
8.2.4	<i>Voorwaardelijke compilatie</i>	324
8.2.5	<i>Expliciet aanroepen van een constructor</i>	325
8.3	<i>Operator overloading</i>	326
8.3.1	<i>Unaire, binaire en ternaire operatoren</i>	326
8.3.2	<i>Een somfunctie</i>	328
8.3.3	<i>Tijdelijk object door een constructor laten maken</i>	330
8.3.4	<i>Een operator + in plaats van een somfunctie</i>	330
8.3.5	<i>Welke operatoren mag je overladen?</i>	333
8.3.6	<i>Overladen van unaire en binaire operatoren</i>	333
8.3.7	<i>Overladen van een toekenningsoperator</i>	334
8.3.8	<i>De pointer this</i>	335
8.4	<i>Globale en friend-operatoren</i>	337
8.4.1	<i>De friend-operator*()</i>	338
8.4.2	<i>Implementatie van friend buiten de klasse</i>	340
8.4.3	<i>Een globale operatorfunctie</i>	341
8.4.4	<i>Een insertion- of uitvoeroperator</i>	342
8.5	<i>Conversie van klasse naar een standaardtype</i>	343

8.6	Conversie tussen klassen	344
8.7	Vragen	348
8.8	Opgaven	349
9	Overerving	353
9.1	Inleiding	353
9.2	Een basisklasse	353
9.3	Afgeleide klasse	355
9.3.1	<i>Private, public en protected</i>	356
9.3.2	<i>Defaultconstructor van de basisklasse</i>	358
9.3.3	<i>Eigen constructor voor afgeleide klasse</i>	358
9.4	Functie-overriding	360
9.5	Generalisatie	363
9.5.1	<i>Aanroepen van functie in basisklasse</i>	369
9.6	Afgeleide klasse van een afgeleide klasse	370
9.6.1	<i>Initialisatielijst en indirecte basisklasse</i>	373
9.6.2	<i>Wat erft een afgeleide klasse niet?</i>	373
9.6.3	<i>Geërfde constructor (inherited constructor)</i>	373
9.7	Toegangsregels	374
9.7.1	<i>Wanneer gebruik je wat?</i>	375
9.8	Multiple inheritance	376
9.8.1	<i>Ambigüiteit bij multiple inheritance</i>	378
9.9	Virtuele basisklasse	380
9.9.1	<i>Oplossing via virtuele basisklasse</i>	383
9.9.2	<i>Initialisatielijst en virtuele basisklassen</i>	386
9.10	Samenvatting	386
9.11	Vragen	387
9.12	Opgaven	387
10	Dynamisch geheugen	391
10.1	Inleiding	391
10.2	Dynamische arrays	391
10.3	Een destructor	394
10.3.1	<i>Wanneer wordt een destructor aangeroepen?</i>	396
10.3.2	<i>Wanneer moet je zelf een destructor schrijven?</i>	399
10.3.3	<i>Memory leakage</i>	399
10.4	Smart pointers	400
10.4.1	<i>De klasse weak_ptr</i>	402
10.5	Geheugentekort en new	403
10.6	Copy-constructor en toekenningsoperator	404
10.6.1	<i>De copy-constructor</i>	405
10.6.2	<i>Een diepe kopie met de copy-constructor</i>	406
10.6.3	<i>De toekenningsoperator</i>	407
10.6.4	<i>Verwijderen van copy-constructor en toekenningsoperator</i>	411
10.6.5	<i>Herdefinitie van de indexoperator []</i>	412

10.7	Objecten en dynamisch geheugen	414
10.8	Een lineaire lijst	416
10.8.1	<i>De opbouw van de lijst</i>	417
10.8.2	<i>Het maken van de lijst</i>	417
10.8.3	<i>De lijst langslopen met een pointer</i>	419
10.9	Een verbeterde lijst	421
10.9.1	<i>Een friend-klasse</i>	422
10.9.2	<i>Geen friend, maar public get-functies</i>	424
10.9.3	<i>Een destructor voor de lijst</i>	425
10.10	Een iterator voor een lijst	427
10.11	Samenvatting	432
10.12	Vragen	432
10.13	Opgaven	433
11	Templates	437
11.1	Inleiding	437
11.2	Functietemplates	437
11.2.1	<i>Een template voor de functie maximum</i>	438
11.2.2	<i>De essentie van het templatemechanisme</i>	441
11.2.3	<i>Functietemplate en zelf gedefinieerde klasse</i>	441
11.2.4	<i>Een functietemplate met twee template-argumenten</i>	443
11.2.5	<i>Overladen van functietemplate</i>	444
11.2.6	<i>Schrijven van een functietemplate</i>	445
11.3	Klassentemplates	445
11.3.1	<i>Implementatie van lidfunctie van een templateklasse</i>	448
11.3.2	<i>Defaultwaarde voor een template-argument</i>	448
11.4	Template voor een lineaire lijst	448
11.4.1	<i>Generieke lineaire lijst</i>	449
11.4.2	<i>Lidfuncties buiten de templateklasse implementeren</i>	452
11.5	Afgeleide klasse van een klassentemplate	452
11.5.1	<i>Afgeleide klasse zonder genericiteit</i>	453
11.5.2	<i>Afgeleide klasse met behoud van genericiteit</i>	456
11.6	De klasse list uit de standaardbibliotheek	456
11.6.1	<i>De functie merge()</i>	462
11.6.2	<i>Een templatefunctie voor print()</i>	464
11.7	De klasse stack	466
11.7.1	<i>Nut van een stack</i>	467
11.8	De klasse queue	469
11.9	De klasse deque	471
11.10	Samenvatting	472
11.11	Vragen	472
11.12	Opgaven	473
12	Algoritmen	479
12.1	Inleiding	479

12.2	Soorten iterators	479
12.2.1	<i>Speciale iterators</i>	480
12.3	Een algoritme	481
12.3.1	<i>Een const-iterator</i>	483
12.3.2	<i>Een algemenere versie van zoek()</i>	484
12.3.3	<i>Een functie zoek() voor een vector</i>	485
12.3.4	<i>Een templateversie van zoek()</i>	486
12.3.5	<i>De laatste versie van zoek()</i>	488
12.4	Het algoritme <code>std::find()</code>	490
12.5	De algoritmen in de namespace <code>std::ranges</code> en in <code>std</code>	490
12.6	Het algoritme <code>for_each()</code> uit <code>std::ranges</code>	491
12.6.1	<i>for_each met een globale functie</i>	491
12.7	Functieobjecten	494
12.7.1	<i>Het algoritme for_each en een functieobject</i>	496
12.7.2	<i>Functieobject met attribuut</i>	498
12.8	Lambdafuncties	499
12.8.1	<i>Lambdafunctie met of zonder expliciet return type</i>	501
12.8.2	<i>Lambdafunctie met capture list</i>	502
12.8.3	<i>Capture by reference</i>	503
12.8.4	<i>Mogelijkheden voor de capture list</i>	505
12.8.5	<i>De voorziening mutable</i>	505
12.8.6	<i>De lay-out van de code van een lambdafunctie</i>	506
12.9	Projection	506
12.10	Unair predicaat	509
12.10.1	<i>Unair predicaat en find_if()</i>	510
12.10.2	<i>Projectie en find_if</i>	512
12.11	Sorteren	513
12.11.1	<i>Objecten sorteren met behulp van een predicaat</i>	515
12.11.2	<i>Objecten sorteren met behulp van een projectie</i>	518
12.11.3	<i>Objecten sorteren met de spaceship-operator <=></i>	519
12.12	Het algoritme <code>copy()</code>	521
12.12.1	<i>Een back inserter</i>	522
12.12.2	<i>Een front inserter</i>	523
12.12.3	<i>Een inserter die gebruikmaakt van insert()</i>	523
12.12.4	<i>Inhoud van container naar het scherm met copy</i>	524
12.13	Een bestand schrijven en lezen met <code>copy()</code>	524
12.13.1	<i>Een bestand maken met copy()</i>	525
12.13.2	<i>Een bestand lezen met copy()</i>	527
12.13.3	<i>Bestand met gehele getallen direct op het scherm zetten</i>	528
12.13.4	<i>Alle karakters uit een bestand lezen</i>	528
12.14	Zelfgedefinieerd type en <code>copy()</code>	529
12.15	De algoritmen <code>iota()</code> en <code>transform()</code>	531
12.16	Over algoritmen, iterators, containers en streams	534
12.17	Ranges en views in C++20	534
12.17.1	<i>Andere views</i>	536

12.17.2	<i>De adapters drop, take en filter</i>	538
12.17.3	<i>De view iota</i>	539
12.18	Verschillende notaties van views en ranges	541
12.19	Samenvatting	542
12.20	Vragen	543
12.21	Opgaven	543

online boek 13 Polymorfie en virtuele functies

13.1	Inleiding
13.2	Drie klassen met een functie die <code>to_string()</code> heet
13.3	Een virtuele functie
13.3.1	<i>Polymorfie</i>
13.4	Polymorfie met rechthoeken en driehoeken
13.4.1	<i>Dynamische of late binding</i>
13.5	Abstracte klasse
13.5.1	<i>Abstracte klasse Figuur</i>
13.5.2	<i>Een vector en een lijst met figuren</i>
13.6	Virtuele functies in een keten van afgeleide klassen
13.7	Polymorfie via een referentie
13.8	Samenvatting
13.9	Vragen
13.10	Opgaven

online boek 14 Streams

14.1	Inleiding
14.2	Standaardstream-objecten
14.2.1	<i>De standaardstreams cerr en clog</i>
14.3	Een bestand maken en lezen met behulp van een stream
14.3.1	<i>Een bestand lezen met behulp van een stream</i>
14.3.2	<i>Strings en bestanden</i>
14.4	Het controleren van een stream
14.4.1	<i>De functies eof(), fail(), bad() en good()</i>
14.5	Drie manieren om een tekstbestand te lezen
14.6	I/O met objecten
14.6.1	<i>I/O van objecten met behulp van operatoren</i>
14.6.2	<i>Minder foutgevoelige invoer</i>
14.7	Objecten in een bestand
14.8	Mogelijkheden bij het openen van bestanden
14.8.1	<i>Bestand later aan een stream koppelen</i>
14.8.2	<i>Bestand openen om aan het einde toe te voegen</i>
14.8.3	<i>Bestaand bestand openen en leegmaken</i>
14.8.4	<i>Per se werken met een bestaand bestand</i>
14.8.5	<i>Per se niet werken met een bestaand bestand</i>
14.8.6	<i>Lezen én schrijven uit een bestand: fstream</i>
14.9	Random file access

- 14.9.1 *Bestand openen met de file-pointer aan het eind*
- 14.9.2 *Opnieuw lezen uit hetzelfde bestand: de functie clear()*
- 14.10 Binaire bestanden
 - 14.10.1 *Maken van een binair bestand*
 - 14.10.2 *Lezen van een object uit een binair bestand*
- 14.11 Kopiëren van een bestand
 - 14.11.1 *Kopiëren met behulp van rdbuf()*
 - 14.11.2 *Kopiëren van een bestand naar het beeldscherm*
- 14.12 Schrijven naar een string
 - 14.12.1 *Een ostreamstring leegmaken*
- 14.13 Vragen
- 14.14 Opgaven

online boek 15 Excepties

- 15.1 Inleiding
- 15.2 Try, throw en catch
- 15.3 Verschillende excepties van hetzelfde type
- 15.4 Excepties van een verschillend type
 - 15.4.1 *Exceptie definiëren binnen een klasse*
 - 15.4.2 *Alle excepties opvangen*
 - 15.4.3 *Informatie in de exceptie*
- 15.5 Re-throw
- 15.6 Excepties als afgeleide klassen
- 15.7 Standaardexcepties
 - 15.7.1 *Zelf een standaardexceptie opwerpen*
- 15.8 Excepties specificeren in de declaratie van een functie
- 15.9 Invoer van bepaald soort getallen
- 15.10 Vragen en oefeningen
- 15.11 Opgaven

online boek Bijlage A: Header en implementatie

- P.1 Inleiding
- P.2 Klassendeclaratie in header file
- P.3 Implementatie in .cpp-bestand
 - P.3.1 *Gebruikmaken van Persoon*
- P.4 Dubbele declaraties voorkomen
- P.5 Templates en headers

online boek Bijlage B: Namespaces

- Q.1 Inleiding
- Q.2 Een namespace maken
- Q.3 Een namespace gebruiken
- Q.4 De standaardbibliotheek en std

online boek

Bijlage C: Tabel van operatoren

online boekBijlage D: Gereserveerde woorden (keywords) en
voorgedefinieerde identifiers

Index

547

Voorwoord bij de zevende druk

Deze zevende editie van *Aan de slag met C++* is geheel herzien.

Aanleiding voor de herziening is de voortdurende ontwikkeling van de taal C++. Sinds 2011 vinden om de drie jaar updates van C++ plaats. Een aantal van deze wijzigingen, voor zover relevant voor de in dit boek besproken onderwerpen, zijn in deze nieuwe editie opgenomen.

Ik heb in deze editie het gebruik van *using namespace std* vermeden, maar benoem waar nodig de voorzieningen uit de standaard library expliciet, zoals in *std::cout* of eventueel *using std::cout*.

In het algemeen heb ik uniforme initialisatie toegepast; dit maakt het onderscheid tussen initialisatie en assignment gemakkelijker.

De grootste wijzigingen zijn te vinden in hoofdstuk 12, dat in zijn geheel is herschreven en waarin, naast de standaardalgoritmen, ook aandacht is voor ranges en views uit de nieuwe *std::ranges* library.

Net als in de vorige editie eindigt elk hoofdstuk met een samenvatting, vragen en programmeeropgaven. De code van veel voorbeelden uit het boek, de antwoorden op de vragen en uitwerkingen van de meeste opgaven zijn te vinden op www.aandeslagmetcpp.nl. Verder is hier het online boek te raadplegen met de extra hoofdstukken 13, 14 en 15. De bijlagen zijn ook op de website te vinden. De code van alle genummerde voorbeelden, evenals die van de uitwerkingen van de opgaven, is getest met de g++-compiler, versie 10.1 met compileroptie `std=c++20`.

Op www.cppreference.com kun je veel informatie vinden over C++ en de standaardlibrary.

Gertjan Laan
Zaandam, voorjaar 2021
www.gertjanlaan.nl



1



1.1 Inleiding

Deze editie is geschreven met een lezer in gedachten die enigszins bekend is met elementaire begrippen en principes van veelgebruikte programmeertalen. Die kennis heb je bijvoorbeeld opgedaan door code te schrijven in Java, HTML en JavaScript, of Python. Hoewel veel programmeertalen in de basis op elkaar lijken, kunnen ze in die basis ook op subtiele wijze van elkaar verschillen. Subtiliteiten in een programmeertaal kunnen essentieel blijken. Ik geef daarom in dit hoofdstuk een overzicht van de elementaire voorzieningen en eigenaardigheden van C++. Daarbij zal ik steeds proberen de eigenschappen te demonstreren aan de hand van een geschikt, werkend voorbeeld.

1.2 De preprocessor, compiler en linker

Elk C++-programma dat je intikt, de broncode, moet eerst worden vertaald voor het kan worden uitgevoerd. Voor elk platform, zoals Linux, Android, iOS of macOS X en Windows, zijn compilers in omloop die ervoor zorgen dat de vertaalde versie van je programma geschikt is voor dat platform. Het hele vertaalproces verloopt in fasen waarbij achtereenvolgens de volgende onderdelen een rol spelen: de preprocessor, de compiler en de linker.

De preprocessor leest de broncode en gaat al lezend op zoek naar preprocessor-opdrachten, de zogeheten *preprocessor directives*. Een preprocessor directive kun je herkennen aan het hekje # dat ervoor staat. Een voorbeeld van een preprocessor directive is:

```
#include <iostream>
```

<iostream> is de naam van een zogeheten headerbestand (*header file*), dat onderdeel is van het C++-systeem. De complete inhoud van dit bestand moet door de preprocessor worden ingevoegd (geïncluded) in de broncode. In het headerbestand staan definities die noodzakelijk zijn om de compiler zijn werk goed te laten doen. Deze definities behelzen bijvoorbeeld constanten, prototypen en klassen. Hoe die definities eruitzien, wordt in de rest van dit boek duidelijk. Het resultaat van het werk van de preprocessor is een zogeheten *translation unit* (vertalingseenheid).

Na de preprocessor is het de beurt aan de compiler om de translation unit te vertalen. De compiler heeft twee belangrijke taken:

1. Het programma controleren op fouten en daar melding van maken.
2. Als er geen fouten gevonden zijn, het programma vertalen naar machinecode, dat wil zeggen: vertalen naar code die hoort bij de microprocessor die zich in de computer of het apparaat bevindt.

Dit hele proces wordt ook wel aangeduid met *compile-time*. Gedurende *compile-time* kunnen veel fouten tegen de grammaticale regels van de taal worden gevonden. *Compile-time* staat tegenover *runtime* (zie verderop).

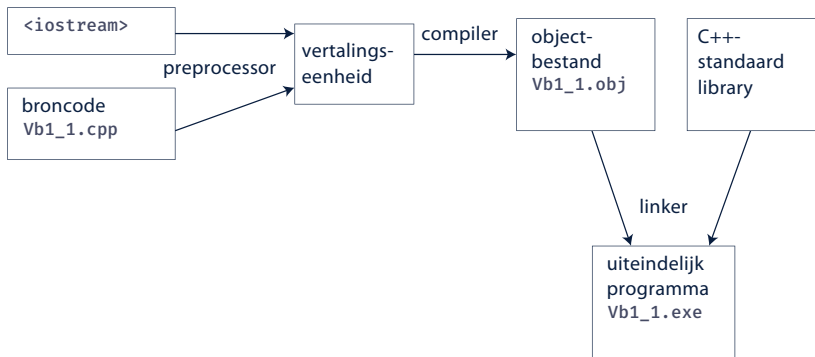
Laten we even aannemen dat het programma taalkundig correct is, zodat er een vertaling tot stand komt. De vertaling wordt eventueel opgeslagen in een tussenbestand waarvan de naam het achtervoegsel `.obj` of `.o` heeft. Deze code heet wel *objectcode*, of doelcode (als tegenhanger van broncode).

De rol van de compiler is nu uitgespeeld. Vervolgens komt de linker aan de beurt. Het woord *linker* komt van het Engelse *to link*, aan elkaar koppelen.

C++ wordt geleverd met honderden voorgedefinieerde functies, klaar voor gebruik. Deze functies zitten opgeborgen in bestanden die *libraries* heten. Het woord *library* betekent letterlijk bibliotheek, maar het is duidelijk dat het hier niet om boeken gaat. De overeenkomst met een echte bibliotheek is dat er een voorraad functies is (in plaats van boeken), waar elk C++-programma naar believen kopieën van kan maken. Alle bibliotheken die standaard bij C++ worden geleverd heten de C++ *standard library*, of C++-standaardbibliotheek.

Een belangrijke taak van de linker is het uit de bibliotheek halen van een kopie van de functies die het vertaalde C++-programma nodig heeft en deze toe te voegen aan het C++-programma. Het resultaat daarvan wordt weggeschreven naar een uitvoerbaar bestand, onder Windows vaak met de extensie `.exe`. Dat uitvoerbare bestand is het bestand waar het allemaal om draait: dit bevat in machinecode alle opdrachten en informatie die het betreffende apparaat en het bijbehorende besturingssysteem nodig hebben om datgene uit te voeren wat je in de broncode van het C++-programma hebt ingetikt. Het proces van het uitvoeren (runnen) van een programma wordt ook wel aangeduid met *runtime*. Ook in runtime kunnen zich fouten voordoen, bijvoorbeeld omdat een berekening in de code verkeerd is opgeschreven, of omdat een bestand waaruit gelezen moet worden niet kan worden gevonden.

In figuur 1.1 is alles nog eens in beeld gebracht.



Figuur 1.1

1.3 Eerste voorbeeld

Laten we eens kijken naar een eenvoudig, maar goed werkend C++-programma:

Voorbeeld 1.1

```

#include <iostream>           //preprocessor directive

int main()                   //start van de functie main()
{
    int a;                   //declaratie van variabele a
    int b;                   //declaratie van variabele b
    int antwoord;           //declaratie van variabele antwoord
    a = 17;                  //a krijgt de waarde 17
    b = 24;                  //b krijgt de waarde 24
    antwoord = a + b;        //antwoord krijgt de waarde van a + b

    std::cout << "Het resultaat is: "; //tekst naar uitvoerscherm
    std::cout << antwoord;           //antwoord naar uitvoerscherm
    std::cin.get();             //wacht op indrukken van de enter-toets
    return 0;                   //zie toelichting
}
  
```

De uitvoer van dit programma ziet er zo uit:

Het resultaat is: 41



1.3.1 Toelichting bij het eerste voorbeeld

Het belangrijkste gedeelte van voorbeeld 1.1 is het gedeelte dat begint met de regel `int main()` en dat helemaal onderaan eindigt met de sluitacolade. Dit gedeelte is een functie die `main()` heet. Elk C++-programma moet een functie hebben met de naam `main()`. Als het programma wordt uitgevoerd, komt altijd als eerste de functie `main()` aan bod. Dit wil zeggen dat de opdrachten die tussen de openings- en sluitacolades achter `main()` staan, als eerste worden uitgevoerd. Deze opdrachten heten ook wel *statements*. Zoals je ziet, eindigt elk statement met een puntkomma.

Als eerste worden in de functie `main()` drie variabelen gedefinieerd: `a`, `b` en `antwoord`. C++ kent veel verschillende typen voor variabelen, deze drie zijn van het type `int`. Het woord `int` is een afkorting van *integer*, wat *geheel getal* betekent. Het benoemen van het type en de naam van een variabele heet ook wel het declareren van die variabele.

In de volgende drie regels krijgen de variabelen een waarde. Dit soort opdrachten heten *assignment-statements* of toekenningsopdrachten. Het teken `=` heet de *assignment-operator* of toekenningsoperator. Met behulp van de assignment-operator geef je een waarde aan een variabele. De variabele staat altijd aan de linkerkant van het teken `=`. Zo'n variabele wordt ook wel een *modifiable lvalue* genoemd. De letter `l` van `lvalue` staat voor *left*.

De in- en uitvoer van gegevens wordt in C++ geregeld via *streams*, ofwel stromen van informatie. Er is bijvoorbeeld een stream van je programma naar het beeldscherm. In voorbeeld 1.1 staan twee opdrachten die met `std::cout` (*cout* spreek je uit als: cie-out) beginnen. Deze regels zorgen voor de uitvoer naar het beeldscherm. Het symbool `<<`, dat uit twee kleinerdantekens bestaat, moet je opvatten als één symbool. Dit symbool heet de *insertion operator* of uitvoeroperator. Met behulp van de uitvoeroperator stuur je iets naar `std::cout`.

Het een-na-laatste statement van het programma is:

```
std::cin.get();
```

Deze opdracht wacht tot je op de entertoets drukt. Sommige ontwikkelomgevingen hebben de gewoonte om het uitvoerscherm in een flits te tonen, te kort om iets van de uitvoer te kunnen zien. Door aan het eind van je programma de opdracht `std::cin.get()` neer te zetten, blijft het uitvoerscherm zichtbaar tot je op Enter drukt. Om ruimte te sparen zal ik in de rest van de voorbeelden in dit boek deze opdracht niet steeds neerzetten, zie ook voorbeeld 1.1a in paragraaf 1.3.2.

In C++ kan een functie een waarde afleveren en de betekenis van `return 0` is dat de functie `main()` de waarde `0` aflevert. In principe kun je deze waarde opvragen vanuit het besturingssysteem. Na afloop van het programma kun je aan de waarde `0` zien dat het programma succesvol is geëindigd. Eventueel kun je het programma een andere waarde dan `0` laten afleveren als er een fout optreedt. Die waarde geeft dan aan dat er iets mis is. Als je als programmeur een lijstje levert

van de mogelijke foutcodes en hun betekenis, kan de gebruiker van je programma nagaan wat er mis is.

Volgens standaard-C++ is het niet noodzakelijk `main()` af te sluiten met `return 0`. Om ruimte te sparen in de voorbeelden zal ik deze opdracht dan ook weglaten, zie ook voorbeeld 1.1a in paragraaf 1.3.2.

Alle namen in C++ zijn opgeborgen in een zogeheten namespace. Een namespace is een verzameling namen die zelf weer een naam heeft. Dit is een manier om naamconflicten te vermijden, die zich snel kunnen voordoen als je met meerdere programmeurs aan een groot project werkt. Een namespace is wat dat betreft vergelijkbaar met een directory. De namespace `std` is een belangrijke namespace. De in voorbeeld 1.1 gebruikte namen `cin` en `cout` maken deel uit van deze namespace, en daarom heten ze voluit `std::cin` en `std::cout`. Meer over namespaces kun je lezen paragraaf 1.4.

Alles wat in de broncode achter twee slashes op dezelfde regel staat, is commentaar of een toelichting voor de menselijke lezer. Het commentaar wordt door de compiler overgeslagen en heeft dus geen invloed op het vertaalde resultaat. Als je meer dan een regel commentaar hebt, kun je dit beginnen met slash sterretje (`/*`) en eindigen met sterretje slash (`*/`). Bijvoorbeeld:

```
/* Dit is ook
   commentaar
   dat 3 regels beslaat */
```

1.3.2 Declaratie van variabelen

In C++ moet je variabelen declareren voordat je ze kunt gebruiken. In plaats van

```
int a;
int b;
int antwoord;
```

mag je ook schrijven:

```
int a, b, antwoord;
```

De typeaanduiding `int` geldt voor alle variabelen die erachter komen, tot aan de eerstvolgende puntkomma. De variabelen moeten van elkaar gescheiden worden door een komma. Als je zo veel variabelen hebt dat ze niet meer op een regel passen, kun je gewoon op de volgende regel doorgaan. De compiler begrijpt ook een schrijfwijze als:

```
int a, b,
    antwoord;
```


Een wat kortere versie van voorbeeld 1.1 zie je in onderstaande code:



Voorbeeld 1.1a

```
#include <iostream>

int main()
{
    int a, b, antwoord;           // declaratie van drie variabelen in een keer
    a = 17;
    b = 24;
    antwoord = a + b;

    std::cout << "Het resultaat is: " << '\n';
    std::cout << antwoord;
}
```

De uitvoer van dit programma is:

```
Het resultaat is:
41
```

In dit geval komt de uitvoer op twee verschillende regels, omdat '\n' ervoor zorgt dat de cursor in de uitvoer naar de volgende regel gaat. '\n' is een zogeheten *escape character*, zie ook paragraaf 1.8.2.

De naam van een variabele heet ook wel een *identifier*. Niet elke identifier is geschikt als naam voor een variabele. Namen van variabelen in C++ moeten voldoen aan de volgende regels:

- Een naam mag je opbouwen uit kleine letters, hoofdletters, cijfers en de *underscore* _.
- Een naam mag nooit beginnen met een cijfer.
- Een naam mag niet een gereserveerd woord (*reserved word*) of een voorgedefinieerde identifier (*predefined identifier*) zijn.

Een gereserveerd woord is een woord dat in de taal C++ een speciale betekenis heeft. Een voorgedefinieerde identifier is een naam die al eens gebruikt is, maar niet tot de taal zelf behoort. Gereserveerde woorden heten ook wel *keywords*. Een lijst van keywords vind je in bijlage D.

Namen die aan de genoemde regels voldoen zijn bijvoorbeeld:

X	x	a23
Hoogte	ANNA	_a23
Hoogte	breedte1	x_1
x1	breedte_1	x_2
x2	Dit_is_een_naam	aantalPersonen

```

2a                // begint met een cijfer
Dit is geen naam // bevat spaties
d'66              // bevat apostrof
4711             // begint met een cijfer
C&A              // bevat & (ampersand)
C++              // bevat plustekens

```

C++ is *case sensitive* en maakt dus verschil tussen hoofdletters en kleine letters: `hoogte` is een andere variabele dan `Hoogte` (en `HOOGTE` is weer een andere). Je mag namen zo lang maken als je wilt, maar er zijn compilers die onderscheid maken tussen (bijvoorbeeld) alleen de eerste 32 tekens.

Voor de zelfgekozen namen van variabelen in dit boek houd ik mij aan de volgende regels:

- een identifier begint met een kleine letter;
- als een identifier uit twee of meer woorden bestaat, zet je een underscore tussen de woorden. Bijvoorbeeld: `aantal_euros`, `nieuw_saldo`, `percentage_eerste_schijf`;
- identifiers geven bij voorkeur zo precies mogelijk aan wat de betekenis van de variabele is. De naam `temperatuur` of `tijd` is veel duidelijker dan de naam `t`. En `nieuw_saldo` en `oud_saldo` zijn duidelijker dan `saldo1` en `saldo2`.

Er zijn andere goede regels denkbaar dan deze. Zo zijn er veel programmeurs die juist wel underscores gebruiken, `nieuwSaldo` wordt dan `nieuw_saldo`. In elk geval raad ik elke beginnende C++-programmeur sterk aan een keuze te maken en de regels consequent in eigen programma's toe te passen. Hierdoor worden ze – voor jezelf en voor anderen – leesbaarder en begrijpelijker.

1.4 Namespaces

Een namespace is enigszins vergelijkbaar met een folder. Zoals een folder bestanden bevat, bevat een namespace namen. In C++ hebben veel zaken een naam: variabelen, constanten, functies, klassen.

Een van de plezierige dingen van folders is dat je er daardoor niet voor hoeft te zorgen dat alle namen uniek zijn: in twee verschillende folders kun je heel goed twee (verschillende) bestanden hebben met dezelfde naam, bijvoorbeeld `test.cpp`.

Je kunt ze van elkaar onderscheiden door de folder te noemen waar ze in zitten:

```
folder1/test.cpp en folder2/test.cpp
```

of in Windows:

```
folder1\test.cpp en folder2\test.cpp
```

Omdat veel programmeerproblemen overeenkomsten bevatten, hebben programmeurs de neiging vaak dezelfde (Engelse) namen te kiezen: zoals `string`, `list`, `vector`, `count`. Als je tijdens het schrijven van een programma gebruikmaakt van het werk van anderen, bijvoorbeeld van een van de bibliotheken die standaard bij C++ geleverd worden, waar honderden of duizenden namen in zitten, kan het lastig zijn steeds te moeten nagaan of een naam die je zelf bedenkt misschien al in gebruik is. Als dat zo is, klaagt de compiler dat de naam al bestaat: je hebt een *name clash* of *naming collision*: een naamconflict.

De oplossing is de namen onder te brengen in verschillende namespaces. Programmeur a gebruikt bijvoorbeeld namespace `a`, en programmeur b gebruikt namespace `b`. Beide programmeurs zorgen ervoor dat bij definitie van namen deze in hun eigen namespace komen. Als programmeur a een functie `count()` maakt, is de volledige naam (*fully qualified name*) van die functie `a::count()`. En als b ook een functie `count()` maakt, heeft die als volledige naam `b::count()`. Op die manier kun je vrij eenvoudig functies uit verschillende namespaces uit elkaar houden, of ze nu dezelfde naam hebben of niet.

Een belangrijke namespace is `std`, wat een afkorting is van *standard*, deze namespace is onderdeel van de standard library van C++.

De namespace `std` bevat veel namen die je vaak gebruikt, bijvoorbeeld `cout`, `cin` en `string`. Hun volledige naam luidt dan ook: `std::cout`, `std::cin` en `std::string`. Dat betekent dat je in een programma vaak `std::` moet tikken, zoals in voorbeeld 1.1 drie keer. Het intikken van `std::` is op den duur nogal vervelend, zeker als je dat tientallen of honderden keren moet doen. Om dat te voorkomen, kun je vanaf C++17 een `using`-statement gebruiken, zoals in dit voorbeeld:

```
int main() {
    using std::cout, std::cin, std::string;
    ...
}
```

Het statement `using std::cout` wil zeggen dat als je daarna `cout` gebruikt, je dit moet opvatten als `std::cout`. Iets dergelijks geldt voor `cin` en `string`.

Voorbeeld 1.1a komt er met een `using` statement als volgt uit te zien:



Voorbeeld 1.1b

```
#include <iostream>

int main() {
    using std::cout;           // using statement voor std::cout
    int a, b, antwoord;       // declaratie van drie variabelen in een keer
    a = 17;
    b = 24;
    antwoord = a + b;
```

```
cout << "Het resultaat is: " << '\n'; //cout zonder std::
cout << antwoord;
}
```

1.4.1 Over using namespace std

In veel kleinere programma's, en dus in veel leerboeken of in online uitleg, vind je vaak de volgende opdracht bovenaan het programma:

```
using namespace std;

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

Het voordeel hiervan is dat je nu *alle* namen uit `std` kunt gebruiken zonder `std::` te typen. Het grote bezwaar hiervan is dat er erg veel namen in `std` zitten, wat heel makkelijk tot een naamconflict kan leiden. Het idee van namespaces was nu juist om dat te voorkomen.

In principe is het dus beter om `using namespace std` niet te gebruiken. Nu kan dat in een basaal programma van tien regels in een leerboek niet zoveel kwaad: alles is goed te overzien, en een eventuele name clash is snel opgelost. Maar beginnende programmeurs worden gevorderde programmeurs, het aantal regels code kan toenemen tot duizenden, en daarmee de onoverzichtelijkheid over welke namen wel of niet in gebruik zijn. Naamconflicten kun je beter voorkomen. Om die reden zal ik in dit boek `using namespace std` niet gebruiken, maar steeds bij een naam expliciet aangeven uit welke namespace hij komt, zoals in `std::cout`, of met `using` aangeven welke namen uit de namespace gebruikt worden, zoals in `using std::cout;`

1.5 Getaltypen

Naast het type `int` kent C++ nog drie typen voor gehele getallen: `short`, `long` en `long long`. Op veel systemen wordt een getal van het type `short` in twee bytes opgeslagen en heeft dan een bereik van -32768 tot en met 32767. Een `int` wordt doorgaans in vier bytes opgeslagen en heeft een bereik van -2147483648 tot en met 2147483647. Het bereik van `short` is dus veel kleiner dan dat van `int`, maar daar staat tegenover dat het geheugengebruik slechts de helft is. Op veel systemen wordt een `long` in acht bytes opgeslagen, waardoor het bereik van `long` aanzienlijk groter is dan dat van `int`. Je zou verwachten dat een `long long` dan in zestien bytes wordt opgeslagen, maar dat hoeft niet het geval te zijn. De standaard eist slechts dat voor het bereik geldt:

`short ≤ int ≤ long ≤ long long`

Het is dus van belang na te gaan welke waarden worden gehanteerd door de compiler die je gebruikt.

De integer-getaltypen kennen ook een `unsigned`-variant: gehele getallen zonder teken, dus met uitsluitend niet-negatieve waarden. Je krijgt dergelijke typen door het woord `unsigned` voor het getaltype te zetten. Een dergelijk woord heet wel een *type modifier* of kortweg *modifier*. Bijvoorbeeld:

```
unsigned short x;
unsigned int xx;
unsigned long xxx;
unsigned long long xxxx;
```

Als het bereik van een `short` loopt van `-32768` tot en met `32767`, dan omvat het bereik van `unsigned short` de getallen van `0` tot en met `65535`. Wanneer je het negatieve getal `-1` opbergt in een `unsigned short`, wordt de waarde omgezet in `65535`. Het getal `-2` wordt omgezet in `65534` et cetera.

Voor de andere `unsigned`-typen geldt iets dergelijks. Zie voor een overzicht van het bereik van verschillende getaltypen de tabel aan het eind van deze paragraaf. Voor gebroken getallen (getallen met een decimale punt) kent C++ de typen `float`, `double` en `long double`.

Naam	Mogelijk aantal bytes	Kleinste waarde	Grootste waarde	
integer-type				
<code>short</code>	2	<code>-32768</code>	<code>32767</code>	
<code>unsigned short</code>	2	<code>0</code>	<code>65535</code>	
<code>int</code>	4	<code>-2147483648</code>	<code>2147483647</code>	
<code>unsigned int</code>	4	<code>0</code>	<code>4294967295</code>	
<code>long</code>	8	<code>-9223372036854775808</code>	<code>9223372036854775807</code>	
<code>unsigned long</code>	8	<code>0</code>	<code>18446744073709551615</code>	
<code>long long</code>	8	<code>-9223372036854775808</code>	<code>9223372036854775807</code>	
<code>unsigned long long</code>	8	<code>0</code>	<code>18446744073709551615</code>	
floating point-type				Precisie
<code>float</code>	4	<code>3.4e-38</code>	<code>3.4e+38</code>	7 cijfers
<code>double</code>	8	<code>1.7e-308</code>	<code>1.7e+308</code>	15 cijfers
<code>long double</code>	10	<code>3.4e-4932</code>	<code>1.1e+4932</code>	19 cijfers

In veel C++-implementaties kun je in een `float` getallen opbergen van $3.4 \cdot 10^{-38}$ tot $3.4 \cdot 10^{+38}$, zowel positief als negatief. Een `float` wordt weergegeven met een precisie van zeven cijfers. Dat wil zeggen dat de overige cijfers achter de decimale punt worden weggelaten door afronding.

Het type `double` is met 15 cijfers een stuk nauwkeuriger dan `float`.

Een notatie als $2.3 \cdot 10^{+8}$ met een macht van 10 heet ook wel de wetenschappelijke notatie (Engels: *scientific notation*). In C++ schrijf je in plaats van de 10 de letter `e` of `E`. Dus $2.3 \cdot 10^{+8}$ wordt bijvoorbeeld `2.3e+8` of `2.3E8` en $3.15 \cdot 10^{-12}$ wordt `3.15e-12`. In deze tabel staat een overzicht van het aantal bytes en het bereik van getaltypen zoals die in een specifieke C++-implementatie kunnen voorkomen. Bij het floating-pointtype zijn in de tabel bij de kleinste en grootste waarde alleen positieve getallen aangegeven, maar dergelijke grenzen gelden ook in het negatieve gebied. Houd er rekening mee dat de implementatie die je gebruikt kan afwijken van de gegevens in deze tabel. Met de operator `sizeof` kun je het aantal bytes van elk type opvragen, zie voorbeeld 1.2.

Voorbeeld 1.2 Aantal bytes opvragen met `sizeof`



```
#include <iostream>

int main()
{
    using std::cout;
    cout << "short: "      << sizeof(short) << " bytes" << '\n';
    cout << "int: "       << sizeof(int) << " bytes" << '\n';
    cout << "long: "      << sizeof(long) << " bytes" << '\n';
    cout << "long long: " << sizeof(long long) << " bytes"
        << '\n';
    cout << "float: "     << sizeof(float) << " bytes" << '\n';
    cout << "double: "    << sizeof(double) << " bytes" << '\n';
    cout << "long double: " << sizeof(long double)
        << " bytes" << '\n';
}
```

De C++-implementatie die ik momenteel gebruik (g++-10) geeft als uitvoer:

```
short: 2 bytes
int: 4 bytes
long: 8 bytes
long long: 8 bytes
float: 4 bytes
double: 8 bytes
long double: 16 bytes
```

Deze geheel herziene zevende druk van *Aan de slag met C++* is een inleiding in de objectgeoriënteerde programmeertaal C++ volgens de laatste standaarden C++11, C++14 en C++20.

In een heldere, aansprekende stijl zet de auteur eerst de basisprincipes van de taal uiteen. Vervolgens komen onder meer de volgende onderwerpen aan de orde: het ontwerpen van klassen, het werken met objecten, overerving, dynamisch geheugen, templates, polymorfie, streams en exception handling. Daar waar nodig verhelderen UML-klassendiagrammen de broncode.

Verder biedt dit boek een nauwkeurige inleiding in alle belangrijke begrippen uit de standaardbibliotheek (STL) zoals vector, list, stack, queue, deque, iterator, algoritmen, functieobjecten en lambdafuncties.

In de zevende druk is het gebruik van het globale statement 'using namespace std;' zoveel mogelijk vermeden. Uniforme initialisatie wordt vrijwel overal toegepast.

Via de website www.aandeslagmetcpp.nl is aanvullend materiaal beschikbaar. De broncode van alle voorbeelden uit het boek is hier te downloaden. Verder vind je hier het online boek met extra hoofdstukken ter verdieping en diverse bijlagen. Daarnaast zijn de antwoorden op de vragen en uitwerkingen van de opdrachten die aan het eind van elk hoofdstuk staan op de website te raadplegen.

Aan de slag met C++ is geschikt voor programmeercursussen in het informatica onderwijs. Het boek kan ook ingezet worden voor zelfstudiedoeleinden.

Gertjan Laan (www.gertjanlaan.nl) was als docent werkzaam in het hoger onderwijs. Hij gaf jaren les in wiskunde en programmeertalen, waaronder C++ en Java. Hij is auteur van diverse succesvolle boeken, waaronder *Aan de slag met Java en JavaFX*.

www.aandeslagmetcpp.nl
www.boomhogeronderwijs.nl

ISBN 978-90-244-3861-7



9 789024 438617