Programmeermethoden

Controle-structuren

Walter Kosters en Jonathan Vis

week 3: 18–22 september 2023

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

ma	di	wo	donderdag	vrijdag
			11:00–12:45 Gorlaeus zaal 1 college 3 iedereen	~~11:00–12:45 Snellius 303/6/7/9 werkcollege 3, vragenuurtje Wiskundigen~~
			13:15–. . . Snellius 302/3/6/7/9 werkcollege 3, vragenuur iedereen	stiekem gaat het door

DLF: Werkcollege Wiskundigen van vrijdag 22 september $\Rightarrow$ donderdag 21 september 13:15–. . . (laat).

Maandagmiddag 25 september (deadline 18:00 uur): informeel vragenuur in de buurt van de computerzalen

Eerste programmeeropgave

Programmeermethoden

Home Algemeen Materiaal Opdrachten en Cijfers

Opdrachten en Cijfers

Programmeermethoden 2023
Eerste programmeeropgave: Vermenigvuldiging

De *eerste* programmeeropgave van het vak Programmeermethoden in het najaar van 2023 heet *Vermenigvuldiging*; 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 de kandidaat weten op welke dag hij/zij geboren is. En als je niets van vermenigvuldigen 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 *25 september 2023* 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!

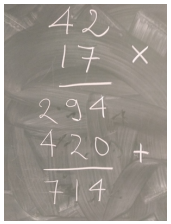
Er wordt gekeken of de aanstaande student enigszins kan vermenigvuldigen. Wiskundig inzicht is namelijk vereist voor een beta-studie. Mocht dat niet zo zijn, wordt er getest hoe het met de kunst- of literatuurkennis staat.

De gebruiker moet allereerst het product van twee willekeurige gegeven positieve gehele getallen schatten, zeg 42 en 17. Beide getallen moeten tussen 10 en 99 liggen, grenzen inbegrepen. De gebruiker krijgt, of diens antwoord goed of fout is, de bekende vermenigvuldiging te zien. In ons voorbeeld:

```
  42
  17 x
  --
 294
 420 +
 ---
 714
```

De getallen en streepjes moeten verticaal goed uitgelijnd staan. Als een rij 0 zou worden (bijvoorbeeld bij 42 maal 20) bestaat het antwoord alleen uit de andere rij: de optelling hoeft dan niet. De tweede rij streepjes (als die er is) heeft drie of vier streepjes, afhankelijk van de lengte van het product.

De *verhouding* tussen het gegokte en het juiste antwoord mag hoogstens `EPSILON` (een zekere waarde, een double, bijvoorbeeld 0.1) van 1 verschillen. Anders wordt het fout gerekend. In het voorbeeld (42 maal 17, foutmarge 0.1) moet het antwoord tussen 643 en 785 liggen, grenzen inbegrepen.



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 liefst alleen de eerste 128 gewone karakters.
Bovenaan het programma (in de C++-code dus) staat uiteraard commentaar, waarin een aantal van deze elementen ook weer terugkomen, maar dan meer gericht op programmeurs, bijvoorbeeld de naam van de gebruikte compiler.

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

Uiterste inleverdatum: maandag 25 september 2023, 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 159 van het Snellius.

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?

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

Inleveren: digitaal `bidenharris1.cc` via Brightspace ⇒ Assignments (maak eerst een "Group"); en `mooi.pdf` printen en in de doos bij Snellius 159 doen.

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

`mooi.tex`

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

`bidenharris1.cc`

Mooi printen

Walter Kosters

23 augustus 2023

1 Uitleg

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

2 Tijd

Er is hier veel tijd aan besteed.

Code

En dit is het programma:

```
1
2  // file iets.cc
3  // Dit is een simpel C++-programma, hello world; vermijd overigens regels met meer
       dan 70 karakters
4
5  #include <iostream>
6  using namespace std;
7
8  const double pie = 3.14159; // een constante (of cmath)
9  int main ( ) {
10   double straal;  // straal van de cirkel
11   cout << "Geef straal, daarna Enter .. ";
12   cin >> straal;
13   if ( straal > 0 )
14     cout << "Oppervlakte "
15          << pie * straal * straal << endl;
16   else
17     cout << "Niet zo negatief ..." << endl;
18   cout << "Einde van dit programma." << endl;
19   return 0;
20 }//main
```

`mooi.pdf`

We werken met Donald Knuth's LaTeX, zie de video.

Formules: `$x^{42} - \beta_7$` levert $x^{42} - \beta_7$.

Gebruik Overleaf, zie

korte introductie

En de uitleg bij het derde werkcollege.

Walter Kosters — smallCpp **September 11, 2023**

1 Introduction

In this short note we describe a simple subset smallCpp of the programming language C++ that can be used in first year introductory programming classes at universities. It is intended to be as simple as possible, while still being powerful enough to allow for decent programs. However, we impose several obligations, that are usually optional, e.g., regarding layout. We adhere to the following principles:

- All smallCpp programs can be compiled and run without errors and warnings by `g++`. Of course, this depends on versions and error levels, but we will in some sense be rather strict here.
- We use as few constructs as possible. We will not explain the different constructs in detail here.

Lines in smallCpp programs are at most 70 characters in length. Only characters with ASCII values from 0 up to and including 127 can be used. Only `//` comments can be used, so no `/* ... */` .

2 Types and variables

The only basic types that are allowed are `bool` (with only values `true` and `false`, no 0/1), `char`, `int` and `double`. Notice that for `int` and `double` we do not specify their number of bytes.
We can use single and double arrays. All array bounds must be specified by constants (by the way, no `#defines`), no numbers allowed. No possibility for `new` and `delete` here. In function headings the first `[ ]` must contain exactly one space.
Programmers can define `classes` in the usual way, even with the unfortunate `;` at the end. Only `private` and `public` members are allowed, and all must be specified as such.
Finally, we have pointer types, whose variables can only point to (address) objects of self-defined `classes`. We use `nullptr`, and not `NULL`. The `->` notation must be used to address members. (By exception, `*this` may be used to refer to the whole object itself, if really needed.)
All variables must be declared at the start of the function they are local in (or, of course, in the parameter list of their function); all declarations end with a comment. No global variables are allowed, only global constants.

3 Control structures

We will only use `if`, `while` and `for` statements. (And `switch` in some very special situations.) Also, `for` must be used only in a "simple" way, with all three components. A `do-while` statement is not allowed. Furthermore we have `new` and `delete` to create and destroy objects of self-defined `classes`. (No `delete [ ]`.)
Only `cin` and `cout` may be used to generate in/output from the keyboard and to the screen.
No usage of the ternary operator `...?...:...` is allowed. And no `,`-operator. The operators `++` and `--` are allowed, but only immediately following variables.

4 Layout

First of all, we expect layout to be internally consistent. Programmers use a fixed indentation, say $t = 2$ (or 3, or 4) spaces, and tabs are not allowed. For all statements curly braces `{ ... }` are used, even for single ones. If $k \geq 0$ "open" curly braces have passed, a line starts with exactly $k \times t$ spaces.
Every line contains at most one statement. Multiple declarations/definitions like `int a, b, c;` are not allowed for the moment: all variables need a line of their own, ending with a comment.
A starting `{` is on the same line as its statement/function, and ends that line — apart from optional comment. The concluding `}` is indented in the same way as the statement/function that started it, and is on an empty line (perhaps with comment) by itself, implying that the line starts with $(k-1) \times t$ spaces. Alternatively, the opening `{` is allowed on a line by itself, indented in the same as the corresponding `}`.
Functions are separated by one or more empty lines, and have at least one comment line above their heading.
Binary operators (`<<`, `>>`, `<`, `>`, `==`, `=`, `!=`, `+=`, `-=`, `<=`, `>=`, `+`, `-`, `*`, `/`, `%`, `&&`, `||`; no others are allowed) are always surrounded by at least one space on both sides. In particular this holds for the assigment operator `=`. It also holds for `&` in call by reference notation in function headings, the only place where `&` can be used. When using several `<<`'s, if on different lines, but belonging to the same `cout`, the first ones on a line start at the same indentation; they are not allowed at the end of a line. By the way, `>>` is allowed exactly once after `cin`.

5 Final details

The only keywords that were not mentioned so far, but that can be used, are `#include`, `void`, `main`, `const`, `else` and `return` (in particular: no `goto`, `do`, `static` (for the moment), `struct`, `scanf`, `printf`, `malloc`, ...). And of course those in `using namespace std;` .
When using *casting*, some flexibility is allowed.
We hope that smallCpp will be a fruitful addition to the world of programming.

Example program

```
1
2  // example.cc: a simple smallCpp program
3
4  #include <iostream>
5  #include <cstdlib>  // so rand ( ) will probably work
6  using namespace std;
7
8  const int MAX = 100; // for array length
9
10 // returns maximum of n-element array A
11 int maximum (int A[ ], int n) {
12   int i;             // counter
13   int maxi = A[0]; // for maximum value
14   for ( i = 1; i < n; i++ ) { // start from 1
15     if ( A[i] > maxi ) {
16       maxi = A[i];
17     }//if
18   }//for
19   return maxi;
20 }//maximum
21
22 // main
23 int main ( ) {
24   int B[MAX]; // the array
25   int i;      // counter
26   srand (42); // initialize random number generator
27   for ( i = 0; i < MAX; i++ ) { // fill the array
28     B[i] = rand ( ) % 10000;
29   }//for
30   cout << "Maximum value is " << maximum (B,MAX)
31        << endl;
32   return 0;
33 }//main
```


Zie de richtlijnen voor het programmeren in C++.

C++ kent de volgende controle-structuren:

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

onbekend (maar eindig?) aantal herhalingen

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

“vast” aantal herhalingen `for`

We gebruiken geen labels/goto’s!

Een if-statement gaat informeel als volgt:

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

En een while-statement gaat informeel als volgt:

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

test

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

Let op de accolades om het "compound" (samengestelde) statement. Rond `fietsen (...);` *mogen* accolades — en die gaan we "altijd" zetten.

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

Waar hoort een “else” bij?

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

Overigens: met accolades vermijd je veel problemen!

Waar hoort een "else" bij?

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

Bij de laatste nog "openstaande" if!

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

† vergelijk Occam's razor

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

OK

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

OK++

nog beter: met accolades

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

OK

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

OK (zie ook switch)

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

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

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

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

```
// geen else nodig
```

OK++

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

is equivalent met:

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

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

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

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

Het kan ook zo:

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

Nog enkele voorbeelden van while-loops:

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

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

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

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

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

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

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

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

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

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

Na

$$\overbrace{\texttt{x even?}}$$

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

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

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

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

Er mogen ("moeten") accolades `{ }` rond `x = x / 2;` staan.

Bij een for-loop als

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

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

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

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

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

Een rare for-loop:

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

Dit is hetzelfde als

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

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

Een dubbele for-loop:

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

geeft:

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

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

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

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

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

- maak de eerste programmeeropgave — de deadline is op maandag 25 september 2023, 18:00 uur

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

- lees Savitch Hoofdstuk 2

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

- maak opgaven 6/10 uit het opgavendictaat