Inleiding Programmeren in C++

voor Life Science & Technology

Walter A. Kosters

Leiden Institute of Advanced Computer Science

Universiteit Leiden

januari 2004

Inhoudsopgave

1 Inleiding

Dit dictaat is bedoeld als hulpmiddel om de programmeertaal C++ te leren. Het is verstandig om een goed C++-boek aan te schaffen, met name dat van Ammeraal (zie beneden). De meeste boeken lenen zich enigszins voor zelfstudie, maar er zijn wel enkele extra zaken nodig:

- Ze vertellen vaak niet zoveel over "algoritmen" (rekenvoorschriften). Zo staat er soms bijvoorbeeld geen sorteeralgoritme in.
- Er staat dikwijls niet zoveel in over specifieke compilers, dat wil zeggen over het programma dat onze C++-programma's moet "begrijpen" en uitvoeren.
- Er wordt nauwelijks aandacht geschonken aan specifieke apparatuur.
- Het zijn vaak geen naslagwerken, geen "reference manuals" dus. Daarom is een boek als dat van Deitel en Deitel (zie verderop) wel handig om erbij te hebben—maar het is wel duur.
- En als laatste: uit een boek alleen kun je nooit leren programmeren. Er moet veel programmeer-ervaring opgedaan worden, kortom veel routine!

In dit dictaat staan daarom enkele handleidingen ter aanvulling op de verschillende boeken. Er is een gedeelte gewijd aan "algoritmen": rekenvoorschriften die op een computer kunnen worden uitgevoerd. Aan de hand van voorbeeldprogramma's worden deze toegelicht. Verder staan in het dictaat een groot aantal opgaven en oude tentamens met uitwerking. Op de (werk)colleges worden alles uiteraard systematisch behandeld. Voorkennis voor dit vak is het college Algoritmiek voor biologen; lees ook het bijbehorende dictaat. Via World Wide Web is met bijvoorbeeld `Netscape` of `Explorer` de nieuwste informatie over het vak—en errata voor dit dictaat—te vinden via de pagina

`http://www.liacs.nl/home/kosters/lst/`

Zij die meer willen weten over C++ of over bijvoorbeeld Linux kunnen kijken bij het college Programmeermethoden uit het eerste jaar van de studie Informatica, zie

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

Voor het bij het vak behorende practicum moeten enkele programmeeropgaven in C++ gemaakt worden. Dit mag gebeuren met een C++-compiler naar keuze. We zullen doorgaans werken met Microsoft Visual C++. Dit programma heeft een geïntegreerde omgeving: editor en compiler staan niet meer apart, maar werken samen. Het practicum kan gemaakt worden op de PC's van de universiteit in het Gorlaeus Laboratorium, zalen LCP6 en LCP8. Om op dit systeem te werken heb je een account nodig; als het goed is, hebben alle studenten een dergelijk account. We zullen voor deze programmeeromgeving in dit dictaat enige aanknopingspunten bieden. Liefhebbers moeten thuis maar eens experimenteren met Linux, zie verderop.
Voor zover het C betreft moeten het ANSI-C-programma's zijn (de bekende standaard). Ook voor C++ geldt dat we ons zoveel mogelijk aan deze nieuwe standaard zullen houden. Het is de bedoeling om in C++ te programmeren; nu is vrijwel elk C-programma automatisch een C++-programma (omgekeerd niet), maar we zullen toch proberen een C++-stijl

aan te leren. Om een voorbeeld te noemen: in C++ zet je een tekstje, zoals Blablabla, op het beeldscherm met `cout << "Blablabla";`, terwijl dit in C *moet* (en in C++ dus ook *zou mogen*) met `printf("Blablabla");`. Wij gebruiken dus `cout`. De meeste editors zijn behulpzaam bij het handhaven van een consequente layout en programmeerstijl.
Er zijn vele andere manieren om iets meer over C++ te weten te komen. In de boekwinkels liggen tientallen titels, waaronder:

- L. Ammeraal, Basiscursus C++, Academic Service, 1999 (derde herziene uitgave). Goed(koop) boek. Met name de hoofdstukken 1 tot en met 5 zijn van belang voor dit college.
- W. Savitch, Absolute C++, Addison-Wesley, 2002. Goed boek.
- H.M. Deitel en P.J. Deitel, C++ how to program, fourth edition, Prentice Hall, 2003. Zeer uitgebreid en duidelijk (maar ook prijzig), met Standard Template Library en UML.
- D. Chapman, Teach yourself Visual C++ 6 in 21 days, Sams Publishing, 1998. Als je Windows-applicaties wilt maken.

Hiernaast bestaan vele elektronisch hulpmiddelen: via WWW (via het al eerder genoemde Internet dus) valt er genoeg te vinden.
Hoe moet dit dictaat nu gebruikt worden? Vanzelfsprekend wordt op de (werk)colleges hierover het nodige gezegd. De hoofdstukken met algoritmen en opgaven sluiten direct aan op de (werk)colleges. Het verstandigste is om zo snel mogelijk de hoofdstukken 1 en 2 uit het boek van Ammeraal door te lezen, en zo spoedig mogelijk zelf aan de gang te gaan met het schrijven van programma's.
Aan dit dictaat hebben velen op allerlei wijze meegeholpen; we willen hen allen—zonder namen te noemen—hier hartelijk danken. Op- en aanmerkingen over het dictaat kunnen via email gestuurd worden aan: `kosters@liacs.nl` .

Walter A. Kosters, Opleiding Informatica — Universiteit Leiden, januari 2004.

2 C++ op een PC

We behandelen achtereenvolgens Visual C++, Dev-C++ en Linux. Alleen Visual C++ is niet gratis (er zijn wel gratis demo-versies); het programma is geinstalleerd op de computers in de computerzalen LCP 6 en 8.

2.1 Visual C++

Start Visual C++ door in het Start-menu bij Programma's aan te klikken (even zoeken in de submenu's): Microsoft Visual C++ 6.0, en daarbinnen nogmaals Microsoft Visual C++ 6.0. Sluit de "Tip of the Day". Selecteer in het **File**-menu: **New** (en een volgende keer **Open** om een reeds bestaande file aan te passen), en dan bij **Files** (klik eventueel eerst op het tabblad): **C++ Source File** en dan OK. Tik in het grote tekstwindow een klein programma in, bijvoorbeeld een "Hello world" programma zoals

```
#include <iostream>
using namespace std;
int main ( ) {
  cout << "Hier ben ik ..." << endl;
  return 0;
} // main
```

en save dit als bijvoorbeeld `poging.cpp`. De afspraak is dat C++-programma's als extensie `.cpp` of `.cc` hebben. Het saven kan bij het **File**-menu, en dan uiteraard de optie **Save**. Let er goed op waar de file `poging.cpp` gesaved wordt, en zet de file aan het eind van je sessie even op een eigen diskette—of mail hem aan een van je eigen emailadressen. Voor de meeste commando's zijn trouwens ook ikonen en korte toets-combinaties te gebruiken. Nu bij **Build** *compileren*. Er wordt eerst gevraagd een bijbehorend project te openen; doe dat! Laat dan compileren (dat is eigenlijk "vertalen" naar een dichter bij de computer zittende taal; via **Compile** `poging.cpp`), daarna **Build** `poging.exe` (dat verzorgt het *linken*, dat wil zeggen het aan elkaar vastknopen van allerlei reeds vertaalde programma's, waaronder standaardbibliotheken) en tot slot **Execute** `poging.exe` (dat voert het programma uit). Het programma wordt nu in een soort DOS-window gedraaid. Als er fouten gevonden zijn, dan kun je deze in het onderste window zien. Door met de rechter muisknop zo'n fout aan te klikken, kom je met iets als "Go to error" snel op de juiste plek in het C++-programma terecht; of liever gezegd: in de buurt van de gemaakte fout. Verander maar eens in het voorbeeldprogramma `cout` in `cut`, en compileer opnieuw.
Een volgende keer kun je de file benaderen via de optie **Open** van het **File**-menu (en dan `poging.dsw`, het project, binnenhalen), of sneller nog via de optie **Recent Workspaces** uit het **File**-menu. Opmerking: door in het **Build**-menu bij **Set Active Configuration** de optie **Debug** in **Release** te veranderen worden veel kleinere executeerbare files gefabriceerd.
In tientallen boeken wordt stap voor stap in alle details uitgelegd hoe je fraaie Windows95/98/NT/2000/ME/XP/... applicaties kunt maken, met windows met knoppen en dergelijke. Hoe het gaat met projecten wordt daarin ook onthuld.

Tot slot. Laten we aannemen dat het "Hello world" programma gelukt is. Hoe nu verder? Probeer aan de hand van Hoofdstuk 1 uit het boek van Ammeraal en Hoofdstuk 3.1 uit dit dictaat wat boeiender programma's te schrijven.

Enkele suggesties:

- Een programma dat aan de gebruiker twee getallen vraagt, en als resultaat hun som afdrukt.
- Idem, maar nu mag de gebruiker ook nog kiezen of de som of het product wordt afgedrukt.
- Aan de gebruiker wordt een temperatuur in graden Celcius gevraagd, en het programma drukt de temperatuur in graden Fahrenheit af (of andersom), zie Opgave 2 van Hoofdstuk 4 van dit dictaat.

Lees daarna Hoofdstuk 2 uit het boek van Ammeraal door, etcetera.

2.2 Dev-C++

Een zeer goede *gratis* programmeeromgeving is Dev-C++, te verkrijgen via

`http://www.bloodshed.net/`

Deze lijkt wel wat op Visual C++, maar heeft aanzienlijk minder toeters en bellen.

2.3 Linux

Voor diegenen die het operating systeem *Linux* eens willen proberen bestaat tegenwoordig een erg eenvoudige mogelijkheid. Haal namelijk (met een breedbandverbinding) een gratis *Knoppix*-CD op van `http://www.knoppix.org/` . Via deze website is een "ISO-image" van de CD te krijgen, die je zelf op een eigen CD moet branden.
Met deze CD moet je vervolgens je PC opstarten; zorg er wel voor dat bij het opstarten eerst op de CD gekeken wordt. Na enige tijd wachten kom je terecht in het windowsysteem KDE. Hierin kun je een console-window (ook wel terminal of shell genoemd, hoewel het eigenlijk allemaal wat verschillende zaken zijn) openen door op de ikoon met de schelp te klikken — of via allerlei menu's. In zo'n window kun je UNIX-commando's intikken, bijvoorbeeld `ls` , dat een overzicht over je files geeft.
Een eenvoudige editor is *nedit*: met het commando `nedit poging.cc &` edit je de file `poging.cc` (denk aan de `&`). Tik hier een eenvoudig C++-programma in. Je kunt de file daarna vanuit de shell compileren met `g++ -Wall -o poging poging.cc` , en tot slot runnen met `./poging` . Hopelijk volstaan deze uiterst summiere opmerkingen. Voor meer informatie raadplege men de talloze andere bronnen.
Knoppix heeft de eigenschap dat "alles" in het RAM-geheugen van de computer terecht komt, ook je zelfgemaakte files. Wil je deze echt bewaren, zet ze dan op een floppy of de harde schijf — wat in Linux de eerste keer misschien iets meer moeite kost dan je gewend bent.
Linux is een bijzondere wereld, met merkwaardige bewoners. Liefhebbers kunnen zich geheel uitleven met de talloze zeer uitgebreide versies, zoals Mandrake, SuSe en RedHat.

3 Algoritmen

In dit gedeelte van het dictaat zullen we diverse voorbeeldalgoritmen (kort) behandelen. Steeds wordt de C++-code gegeven, en daarbij enige uitleg geboden. Relevante delen uit het boek van Ammeraal worden bekend verondersteld.

3.1 Een eerste programma

Als voorbeeld bekijken we het volgende C++-programma:

```
// Dit is een regel met commentaar ...
#include <iostream>          // moet bovenaan ieder C++-programma
using namespace std;

const double pie = 3.14159;  // een constante (beter uit cmath)

int main ( ) {
  double straal;  // straal van de cirkel
  cout << "Geef straal, daarna Enter .. ";
  cin >> straal;
  if ( straal > 0 )
    cout << "Oppervlakte " << pie * straal * straal << endl;
  else
    cout << "Niet zo negatief ..." << endl;
  cout << "Einde van dit programma." << endl;
  return 0;
}  // main
```

De eerste regel van dit programma bevat commentaar voor de menselijke lezer. In de tweede en derde regel wordt verteld dat er zaken uit de "bibliotheek" `iostream` (in iets oudere C++-versies ook wel `iostream.h`) gebruikt zullen gaan worden; ontbreekt deze regel, dan zijn C++-woordjes als `cout` opeens onbekend voor de compiler die dit programma moet "begrijpen". Dan volgt een regel waarin aan de naam `pie` het getal `3.14159` gekoppeld wordt; dit is een constante, die verderop in het programma niet meer gewijzigd kan en mag worden.
Het eigenlijke C++-programma begint bij de regel met `main`. Eén voor één worden dan de regels (of beter statements, de opdrachten) uitgevoerd. Eerst wordt een variabele met de naam `straal` aangemaakt; de betreffende geheugenruimte kan een reëel getal—of beter gezegd een benadering daarvan—bevatten. Dan komt er een tekst op het beeldscherm via de `cout`-regel. Alles wat tussen de `"`'s staat, komt letterlijk op het scherm. Dankzij de `cin`-regel wordt de door de gebruiker ingevoerde waarde gestopt in de variabele `straal`. Dan komt een `if`-statement, dat verscheidene regels beslaat. Indien `straal` een waarde groter dan `0` bevat, wordt de oppervlakte van de betreffende cirkel uitgerekend en afgedrukt. Merk op dat wanneer de `"`'s ontbreken, de waarde van een uitdrukking (expressie) wordt afgedrukt door een `cout`-statement, maar met `"`'s erbij wordt letterlijk wat er tussen de `"`'s staat op het scherm gezet—inclusief spaties en hoofdletters. In één `cout`-statement kunnen ook diverse zaken, gescheiden door `<<`'s, afgedrukt worden. Indien `straal` een waarde kleiner dan of gelijk aan `0` bevat, wordt de tekst `Niet zo negatief ...` op het

scherm gezet, waarbij dankzij de `endl` de cursor naar een nieuwe regel gaat; de `endl` zorgt er trouwens ook voor dat de "output-buffer" op het beeldscherm geleegd wordt: hij "flusht". Tot slot wordt nog een tekst afgedrukt, waarna het programma klaar is: met `return 0;` wordt het beeindigd.
Zou een van de laatste regels uit het programma zo ingesprongen zijn dat de twee `cout`-regels direct onder elkaar staan, dan maakt dat niets uit voor de werking. Er wordt wel een ander effect bereikt door een `{` voor de op een na laatste `cout`-regel en na de laatste `cout`-regel te zetten: deze horen dan bij elkaar en worden als één statement opgevat. De laatste `cout`-regel wordt dan niet meer uitgevoerd als `straal` groter dan `0` is.
Denk er aan dat een enkele `=` in C++ een toekenning is, en niet een test op gelijkheid. Zo zou in bovenstaand programma de regel `if ( straal = 0 )` ongeacht de ingevoerde waarde van `straal` resulteren in het uitvoeren van de `else`-situatie. Er wordt namelijk `0` in `straal` gestopt, wat ook meteen het resultaat van de toekening is, en `0` is in C++ hetzelfde als `false`, "niet waar" dus. Testen op gelijkheid gaat in C++ met `==`. In veel andere talen gaat een toekenning overigens met een `:=`.
Een ander probleem is de zogenaamde "hangende else":

```
if ( a > 0 )
  if ( b > 0 )
    cout << "a en b groter dan 0" << endl;
  else
    cout << "???" << endl;
```

Nu is de afspraak dat een `else` hoort bij de laatste nog openstaande `if`, dus in ons voorbeeld bij `if ( b > 0 )`. Wil je hem toch bij de eerste `if` laten horen, dan moeten om het tweede `if`-statement accolades worden gezet:

```
if ( a > 0 ) {
  if ( b > 0 )
    cout << "a en b groter dan 0" << endl;
  }  // if
else
  cout << "???" << endl;
```

Denk erom dat de layout niet alles zegt!
Een schoon scherm/window is overigens als volgt "eenvoudig" te verkrijgen:

```
#include <stdlib>
  ...
  system ("cls");
```

3.2 Loops

Herhalingen worden in C++ met behulp van `for`- en `while`-loops bewerkstelligd. Hoewel ze in zekere zin equivalent zijn, is het een goede programmeerpraktijk om een `while`-loop te gebruiken als het aantal herhalingen van te voren onbekend is ("net zolang zeuren totdat"), of lastig te bepalen, en `for`-loops te reserveren voor situtaties waarbij het aantal herhalingen vast ligt ("drie maal bellen"). Enkele elementaire voorbeelden staan (meteen in functievorm) hieronder. Allereerst:

```
// druk eerste n getallen met hun kwadraat af
void kwadraten1 (int n) {
  int i = 1;  // tellertje, meteen initialiseren
  while ( i <= n ) {
    cout << i << " -- " << i * i << endl;
    i++;
  }  // while
}  // kwadraten1
```

Zolang de waarde van `i` kleiner dan of gelijk aan `n` is, worden de regels tussen de binnenste accolades herhaald. Een `void` functie wordt ook wel "procedure" genoemd. Het is doorgaans verstandig om vanuit functies zo weinig mogelijk uitvoer en invoer te plegen, behalve bij functies die daar speciaal voor bedoeld zijn; zo houd je functies algemeen toepasbaar.

```
// druk eerste n getallen met hun kwadraat af
void kwadraten2 (int n) {
  int i;  // tellertje
  for ( i = 1; i <= n ; i++ )
    cout << i << " -- " << i * i << endl;
}  // kwadraten2
```

De variabele `i` krijgt als startwaarde `1`; zolang de waarde van `i` kleiner dan of gelijk aan `n` is, wordt de `cout`-regel uitgevoerd, waarna telkens `i` met `1` wordt opgehoogd. Zelfs mag je zeggen `for ( int i = 1; i <= n ; i++ )`, waarbij `i` een locale variabele voor de for-loop wordt, die niet nog eens apart in de functie hoeft te worden aangemaakt. Let er wel op dat `i` dan niet buiten de for-loop mag worden gebruikt, iets waar verschillende compilers ook nog wel eens verschillend mee omspringen. Het is verstandig om de test niet `i != n+1` te laten zijn: het levert hier wel hetzelfde resultaat op, maar zou `i` per ongeluk een startwaarde groter dan `n+1` hebben gekregen, dan waren de gevolgen niet prettig.
Merk op dat het tweede programma beter is, immers het aantal doorgangen door de loop ligt vast, en een `for`-loop is dus mooier.
Voor het volgende statement ligt dat anders:

```
while ( x != 1 )
  if ( x % 2 == 0 )
    x = x / 2;
  else
    x = 3 * x + 1;
```

Tot op heden is nog niet bekend of deze `while`-loop voor ieder positief geheel begingetal `x` stopt! En indien het stopt, is het nog maar de vraag wat het aantal doorgangen door de test is geweest. Het probleem staat onder meer bekend als het Syracuse-probleem, het Collatz-probleem of het $3x + 1$-vermoeden.
En bij een programma als

```
x = 1;
while ( x < 1000 )
  x = 2 * x;
```

is het eenvoudig in te zien dat het stopt en dat na afloop `x` de waarde `1024` heeft, maar het aantal doorgangen is in het algemeen—met `n` in plaats van `1000`—een afgeronde logaritme met grondtal 2. Een while-loop is hier superieur.

3.3 Parameters

Als je een rij getallen op grootte wilt sorteren, ligt het voor de hand dat te doen door herhaald geschikte paren getallen onderling te verwisselen. Daar komen we later op terug. We bekijken nu eerst een functies die getallen verwisselt:

```
// Verwissel (swap) de inhoud van a en b.
// We gebruiken hier "call-by-reference" (let op de &).
void wissel (int& a, int& b) {
  int temp;  \\ ook mag (in een keer):
  temp = a;  \\ int temp = a;
  a = b;
  b = temp;
}  // wissel
```

Let er op dat bij een aanroep als `wissel (x,y)` het zo is dat `a` als synoniem voor `x` functioneert: alles wat met de een gebeurt, gebeurt met de ander. In feite wordt er bij aanroep een referentie, een geheugenadres, doorgegeven (*call-by-reference*), in tegenstelling tot het ook uit C bekende *call-by-value*: daar wordt een waarde doorgegeven aan een lokale kopie. Zouden we in het voorbeeld de twee `&`'s weglaten, dan krijgt de locale variabele `a` de actuele waarde van `x` als startwaarde bij aanroep `wissel (x,y)`. Zouden we aanroepen met bijvoorbeeld `wissel (b,a)`, gesteld dat de variabelen `a` en `b` ook op de plek van aanroep bestaan, dan zouden er locale variabelen `a'` en `b'` worden gecreëerd, die als startwaarde de waarde van `b` respectievelijk `a` zouden krijgen. De accenten `'` zijn ter verduidelijking toegevoegd. Een locale variabele als `temp` wordt bij aanroep niet geïnitialiseerd. Overigens noemen we in dit voorbeeld `a` en `b` wel de *formele parameters*, en `x` en `y` de *actuele parameters*. De *globale variabelen* worden buiten en boven de functies aangemaakt, en gelden "overal".
We willen toch nog wat meer stilstaan bij call-by-value en call-by-reference. In een functie als

```
void hoogop (int x) {
  x = x + 10;
}  // hoogop
```

wordt bij een aanroep als `hoogop (y);` allereerst een lokale variabele `x` gemaakt, die meteen geïnitialiseerd wordt op de waarde van de actuele parameter `y`; daarna wordt `x` met `10` opgehoogd en tot slot vernietigd. De functie doet dus eigenlijk niets. Was de eerste regel van de functie `void hoogop (int& x) {` geweest, dan zou de functie gewoon met `y` hebben gewerkt, en zou de waarde van `y` na de functieaanroep `10` hoger zijn geweest dan daarvoor. Dan mag overigens een aanroep als `hoogop (42);` of `hoogop (m+2);` niet meer: tussen de haakjes moet een *l-value* staan, iets waaraan je kunt toekennen en wat dus ook links van de toekenning `=` mag staan. Rechts van toekenningen mogen *r-values* komen: dit kunnen ook expressies zijn als `42` of `m+2`.
Als er "synoniemen" in het spel zijn, lijkt het ingewikkelder:

```
void alias (int r, int & s ) {
  int t;
  t = 3;
  r = r + 2;
  s = s + r + t;
  t = t + 1;
  r = r - 3;
  cout << r << s << t << endl;
}  // alias

int main ( ) {
  int t = 12;
  alias (t,t);
  cout << t << endl;
  return 0;
}  // main
```

De uitvoer van dit programma zal zijn (ga na):

```
11  29  4
29
```

Als in de eerste regel van de functie een `&` voor `r` wordt toegevoegd krijgen we echter:

```
28  28  4
28
```

3.4 Files

Het is eenvoudig om informatie uit files te lezen of naar files weg te schrijven. Het nu volgende programma kopieert een invoerfile onveranderd door naar een uitvoerfile, karakter voor karakter.

```
#include <iostream>
using namespace std;
#include <fstream>
int main ( ) {
  ifstream invoer;
  ofstream uitvoer;
  char kar;
  invoer.open ("invoer.txt",ios::in);  // koppel invoer aan (echte) file
  if ( ! invoer ) {
    cout << "File niet geopend" << endl;
    return 1;  // stopt het programma
  }  // if
  uitvoer.open ("uitvoer.txt",ios::out);
  kar = invoer.get ( );
  while ( ! invoer.eof ( ) ) {
    uitvoer.put (kar);
```

```
    kar = invoer.get ( );
  }  // while
  invoer.close ( );
  uitvoer.close ( );
  return 0;
}  // main
```

Uiteraard kunnen de namen van de files ook in strings worden ingelezen en zo worden doorgegeven. Verder zijn er nog veel meer mogelijkheden met file-IO. Let er wel op dat de functie `eof` pas dan een zinvol resultaat geeft als er een leespoging gedaan is. In het bovenstaande programma schijnt het aantal `get`'s één groter te zijn dan het aantal `put`'s; echter, het afsluitende EOF-symbool wordt door `uitvoer.close ( );` gezet.
Door voor de regel `uitvoer.put (kar);` een test als `if ( kar != 'e' )` te zetten worden alle `e`'s "overgeslagen". Er kan bijvoorbeeld ook op regelovergangen (LineFeed, `'\n'`) getest worden. In een DOS-omgeving worden deze doorgaans onmiddellijk voorafgegaan door een CarriageReturn (`'\r'`) die alleen in "binary mode" (met `ios::binary`) worden gelezen; bovenstaand programma kopieert ze wel goed, maar handelt met één `get` zowel de `'\r'` als de er meteen na staande `'\n'` af.
Ook bij het van `cin` lezen kunnen deze commando's worden gebruikt. Als `cin >>` en `cin.get ( )` of `cin.getline ( )` door elkaar worden gebruikt treden er soms vervelende effecten op: een `\n` (Enter) wordt schijnbaar soms wel en soms niet gelezen. Het is allemaal te begrijpen, maar niet altijd even prettig. Zo slaat `cin >> getal` op zijn jacht naar `getal` bijvoorbeeld spaties en regelovergangen over, en heeft een regelovergang nodig om de buffer van `cin` binnen te krijgen.

3.5 Rekenen met (gehele) getallen

Er zijn talloze eenvoudige problemen met gehele getallen, die een fraaie illustratie vormen voor de mogelijkheden die een programmeertaal biedt. We behandelen er enkele.
Voor die tijd een paar dingen over verschillende types. Voor reële getallen, of liever benaderingen daarvan, heb je in C++ onder meer de types `float` en `double`; een `float` neemt typisch 4 bytes, een `double` typisch 8 bytes. Met *casting* kan een variabele van het ene type in het andere worden omgezet, oftewel "gepromoveerd". Zo wordt door `9/5` altijd, ook al zeg je `x = 9/5` waarbij `x` een `double` is, `1` opgeleverd (want `5` past `1` maal in `9`), maar door `(double)9/5`, en ook door `9.0/5` trouwens, `1.8000`. Nog mooier is de nieuwe C++-notatie: `static_cast<double>(9)`. Om de rest bij deling van `9` door `5` te vinden kan `9 % 5` gebruikt worden; dat levert `4` op. Soms wordt er automatisch "gecast", bijvoorbeeld bij `i = x`, waarbij `x` een `double` en `i` een `int` is; hier wordt naar beneden afgerond en niet naar het dichtstbijzijnde gehele getal, zodat bijvoorbeeld als `x` de waarde `1.9999` heeft `i` de waarde `1` krijgt. In zulke gevallen wordt vaak de truc `i = x + 0.5` gebruikt.
Het afdrukken van "reële" getallen wordt door het volgende voorbeeld hopelijk duidelijk:

```
#include <iomanip>
  ...
  double x = 92.36718;
  cout << "En x is:" << setw (8) << setprecision (2)
       << setiosflags (ios::fixed|ios::showpoint) << x << endl;
```

Met `setw (8)` wordt de *eerstvolgende* af te drukken expressie 8 posities breed rechts uitgelijnd afgedrukt, de "stream manipulator" `setprecision` zorgt er voor dat er voortaan 2 cijfers na de decimale punt/komma komen—het mag ook met de functie `precision`—, en verder zorgt `fixed` er voor dat een decimale punt wordt gebruikt (in plaats van de "scientific notation" als in `5.1e+012`) en laat `showpoint` een getal als `88.00` zo afdrukken, en niet als `88`. Als het goed is verschijnt er nu ␣␣␣`92.37` op het scherm, waarbij ␣ een spatie aanduidt. Er zijn allerlei varianten als `cout << fixed;` mogelijk.

3.5.1 Grootste gemene deler

Allereerst een functie die de *grootste gemeenschappelijke (gemene) deler*, de *ggd*, van twee gegeven gehele getallen berekent.

```
// Bepaal grootste gemene deler (ggd) van gehele x en y
// met behulp van het algoritme van Euclides.
// Neem aan dat x, y >= 0 en dat (x,y) verschilt van (0,0).
int ggd (int x, int y) {
  int rest;
  while ( y != 0 ) {
    rest = x % y;  // rest bij deling van x door y (x modulo y)
    x = y;
    y = rest;
  }  // while
  return x;
}  // ggd
```

Verstokte C-liefhebbers zullen als test overigens schrijven `while ( y ) { ... }`, maar of dat de duidelijkheid bevordert is nog maar de vraag. Als je deze functie al klaar hebt, en later eens een keer gebruikt, doe je aan *bottom-up* programmeren. Ga je de functie pas schrijven wanneer je hem nodig hebt, dan ben je aan het *top-down* programmeren.
Deze functie kan bijvoorbeeld gebruikt worden om breuken te vereenvoudigen:

```
// Vereenvoudig de breuk teller/noemer zoveel mogelijk.
// Neem aan dat teller >= 0 en noemer > 0.
void vereenvoudig (int& teller, int& noemer) {
  int deler = ggd (teller,noemer);
  if ( deler > 1 ) {
    teller = teller / deler;
    noemer = noemer / deler;
  }  // if
}  // vereenvoudig
```

De test `if ( deler > 1 )` hoeft er overigens niet bij. Wel moet er van een hulpvariabele `deler` gebruik gemaakt worden: wordt er geprobeerd twee maal door de grootste gemene deler van teller en noemer te delen (dus tweemaal "dezelfde" functieaanroep, wat op zich ook al niet zo snugger is), dan zal bij de tweede deling de helaas gewijzigde waarde van `teller` gebruikt worden. De noemer van de breuk zal dan niet veranderen, wat—als de ggd niet 1 is—toch in de bedoeling ligt!

3.5.2 Priemgetallen

Er zijn vele manieren om te bepalen of een gegeven geheel getal groter dan 1 een *priemgetal* is, dat wil zeggen geen delers heeft behalve 1 en zichzelf. Het meest voor de hand liggende algoritme—maar lang niet het snelste—is:

```
// Voor sqrt (worteltrekken), vroeger math.h:
#include <cmath>
// Levert true precies als getal een priemgetal is.
bool priem (int getal) {
  int deler = 2;
  double wortel = sqrt (getal);
  bool geendelers = true;
  while ( ( deler <= wortel ) && geendelers ) {
    if ( ( getal % deler ) == 0 )  // deler gevonden: getal niet priem
      geendelers = false;
    deler++;
  }  // while
  return geendelers;
}  // priem
```

De extra hulpvariabele `wortel` voorkomt het steeds opnieuw uitrekenen van de wortel uit het oorspronkelijke getal. Het is duidelijk dat je niet meer voorbij deze wortel hoeft te kijken: als daar een deler van het oorspronkelijke getal zou zitten, zou er ergens voor die wortel ook een moeten zijn.
De while-loop kan overigens ook als for-loop opgeschreven worden. Doorgaans zullen wij, zoals al eerder opgemerkt, alleen een for-loop gebruiken als het aantal keren dat de body doorlopen moet worden van te voren duidelijk bekend is (we moeten iets bijvoorbeeld 17 keer, of n keer, doen). In andere gevallen geven we zoals eerder gezegd de voorkeur aan een while-loop (we moeten dan iets doen net zolang als ...).
Merk trouwens op dat bij de regel met `&&` de tweede test niet meer gedaan wordt als de eerste al uitsluitsel geeft. Een constructie als `if ( ( x != 0 ) && ( 1.0 / x == 0.5 ) )` is dus toegestaan. Dit fenomeen heet *shortcircuiting*. Bij de operator `+` ligt de "evaluatievolgorde" in C++ niet vast: als in een programma ergens `f(x) + g(x)` staat, is niet vastgelegd of eerst `f(x)` of eerst `g(x)` wordt geëvalueerd. Meestal merk je het verschil niet, maar als een functie neveneffecten ("side effects") heeft kan het resultaat van de volgorde afhangen!
Soms is het overigens niet nodig—maar wel duidelijk—al die haakjes te zetten: in C++ is er een zeer precieze prioriteitenlijst van de verschillende operatoren. In de nieuwste C++-versies mag in plaats van `&&` weer `and` gebruikt worden, en analoog `or` voor `||` en `not` voor `!`. Let er op dat condities als `0.0 <= x <= 1.0` niet door middel van `if ( 0.0 <= x <= 1.0 )` worden ingevoerd (dat betekent namelijk iets anders), maar als `if ( 0.0 <= x && x <= 1.0 )`. Iets dergelijks geldt ook voor `if ( x and y > 0 )`, waarschijnlijk wordt in dit geval `if ( x > 0 and y > 0 )` bedoeld.

3.5.3 Random getallen

Vaak is het nodig om willekeurige (*random*) getallen te fabriceren. Daar valt veel over te zeggen, zie bijvoorbeeld de beroemde boeken van Knuth. Daaruit valt te leren dat een

willekeurige methode doorgaans geen willekeurige getallen levert.
Een heel eenvoudige methode is de volgende. Stel je hebt een eerste random-getal, zeg x_{oud} (geheel). We berekenen het volgende random-getal x_{nieuw} als $(a * x_{\text{oud}} + c)$ modulo m, waarbij a, c en m zekere parameters zijn. Dit wordt steeds herhaald, waarbij uiteraard telkens de waarde van x_{oud} aan het begin op de laatste x_{nieuw} gezet wordt. Zo krijgen we schijnbaar willekeurige getallen uit $\{0, 1, 2, \ldots, m-1\}$. Hierbij betekent "modulo" de "rest bij deling door", in C++ een `%`.
Duidelijk is dit niet echt random: door de keuze van a, c, m en de startwaarde van x—ook wel "seed" genoemd—, liggen alle volgende x-waarden vast. We spreken van dan ook van *pseudo-random-getallen*. Voor veel toepassingen is deze methode goed genoeg. Wel is de keuze van de parameters belangrijk. Vaak neemt men $c = 1$, en (als m een macht van 2 is, wat modulo-rekenen eenvoudig maakt) a modulo 8 gelijk aan 5, of (als m een macht van 10 is) a modulo 200 gelijk aan 21. Wordt dit gedaan, dan duurt het lang voordat herhaling optreedt (zodra een x al eerder is geweest, herhaalt zich immers de hele rij!); nu komen alle getallen tussen 0 en $m-1$ aan de beurt. We krijgen dus bijvoorbeeld, met `long` om wat grotere berekeningen toe te staan:

```
// Pseudo-random-getal tussen 0 en 999:
long randomgetal ( ) {
  static long getal = 42;
  getal = ( 221 * getal + 1 ) % 1000;
  return getal;
}  // randomgetal
```

De laatste cijfers van de zo gegenereerde getallen worden hier overigens achtereenvolgens $1, 2, 3, 4, 5, \ldots$—niet zo willekeurig helaas. De initialisatie van een *static* variabele gebeurt maar één keer, terwijl de waarde behouden blijft tussen twee functie-aanroepen. Je kunt hier ook een functie laten aanroepen die van de tijd gebruik maakt, of die een getal aan de gebruiker van het programma vraagt, en zo de randomgenerator initialiseren. Verander niet zomaar die `1000`! Heb je bijvoorbeeld random getallen tussen 3 en 29 (grenzen inbegrepen) nodig, zeg dan `iets = 3 + randomgetal ( ) % 27;` .
In standaardbibliotheken voor C++ zitten meestal verschillende random-generatoren, die overigens doorgaans van het hierboven genoemde type, de *lineaire congruentie methode*, zijn. In `math.h` of `cmath` zijn de betreffende functies meestal te vinden.

3.6 Matrices

Een laatste voorbeeld uit de wereld van de gehele getallen betreft de welbekende driehoek van Pascal. Omdat hier dubbele arrays in voorkomen doen we dat in een aparte paragraaf. Veelvuldig wordt er gerekend met dubbele of 2-dimensionale arrays, beter bekend als *matrices*. Zo'n array, zeg

```
int A[n][n];  // n moet const int n = ... zijn
```

verbeeldt een vierkante `n` bij `n` matrix. Niet-vierkante matrices kunnen uiteraard ook worden gedefinieerd (opgave). Het element uit de `i`-de rij en de `j`-de kolom van een variabele `A` van type `matrix` is te vinden in `A[i][j]`. Denk eraan dat rijen en kolommen beginnen te nummeren met `0`, en eindigen met `n-1`. De misschien wel meest gemaakte fout is onbedoeld buiten de array-grenzen te treden. Omdat arrays in feite pointers naar hun 0-de element zijn, gebruiken dubbele arrays eigenlijk pointers naar pointers!

3.6.1 Driehoek van Pascal

Nu dan de *driehoek van Pascal*. Elk getal in de driehoek is de som van de twee getallen erboven. De getallen heten ook wel *binomiaalcoëfficiënten*. We lopen op een handige manier door de matrix heen en vullen rij voor rij de array-elementen in. We maken gebruik van de bekende identiteit

$$\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$$

en krijgen zo het volgende programma:

```
int main ( ) {
  int i, j;          // voor rijen en kolommen
  int Pascal[n][n];  // in i-de rij, j-de kolom komt "i boven j"
  for (i = 0; i < n; i++)
    Pascal[i][0] = 1;  // de eerste kolom bevat enen
  Pascal[0][1] = 0;
  for (i = 1; i < n; i++) {
    cout << endl << Pascal[i][0] << "  ";  // 1 op een nieuwe regel
    for (j = 1; j <= i ; j++) {
      Pascal[i][j] = Pascal[i-1][j-1] + Pascal[i-1][j];
      cout << Pascal[i][j] << "  ";
    }  // for
    if ( i != n - 1 )
      Pascal[i][i+1] = 0;
  }  // for
  return 0;
}  // main
```

We vullen alleen die array-elementen die we echt nodig hebben:

```
1  0  ...
1  1  0  ...
1  2  1  0  ...
1  3  3  1  0  ...
```

Het is overigens ook mogelijk om met een enkel array te werken. In de `i`-de slag bevat dat array dan de getallen uit de `i`-de rij van de driehoek van Pascal. Nu moet elk getal de som van de twee getallen erboven worden, wat in het enkele array betekent de som van het getal links van jezelf en jezelf. Loop je nu van links naar rechts door het array, dan wordt te vroeg de waarde van de array-elementen gewijzigd, namelijk terwijl je hun oude waarde nog nodig hebt. Door van rechts naar links te lopen, en op te merken dat de driehoek toch symmetrisch is, ontstaat een eenvoudig programma.

```
int main ( ) {
  int rij[n];
  int i, j;
  for (j = 1; j < n; j++)
    rij[j] = 0;
  rij[0] = 1;
```

```
  for (i = 1; i < n; i++) {
    for (j = i; j > 0 ; j--) {
      rij[j] = rij[j-1] + rij[j];
      cout << rij[j] << "  ";
    }  // for
    cout << rij[0] << endl;  // een 1 erachter
  }  // for
  return 0;
}  // main
```

3.6.2 Matrixvermenigvuldiging

Matrices willen graag met elkaar vermenigvuldigd worden. (Voor diegenen die nog niet weten wat dat is: geen nood.) Wij zullen hier "gewoon" twee vierkante matrices, zeg A en B, vermenigvuldigen, met als resultaat C. De definitie is dat

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik} B_{kj}, \quad 0 \leq i, j < n.$$

Zo geldt:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 1 \cdot 5 + 2 \cdot 7 & 1 \cdot 6 + 2 \cdot 8 \\ 3 \cdot 5 + 4 \cdot 7 & 3 \cdot 6 + 4 \cdot 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

De getallen uit de i-de rij van A en de j-de kolom van B worden paarsgewijs vermenigvuldigd en de resultaten opgeteld ten einde het matrixelement C_{ij} op te leveren. In C++:

```
// C wordt A * B (matrixvermenigvuldiging).
void vermenigvuldigen (int A[n][n], int B[n][n], int C[n][n]) {
  int i, j, k;
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++) {
      C[i][j] = 0;
      for (k = 0; k < n; k++)
        C[i][j] += A[i][k] * B[k][j];
    }  // for
}  // vermenigvuldigen
```

Hierbij moeten aanroepen als `vermenigvuldigen (A,A,A)` vermeden worden!
Het doorgeven van meerdimensionale array's aan functies heeft lastige kanten. De volgende functie, die de som van alle array-elementen van het array `A` berekent, maakt een en ander hopelijk duidelijk:

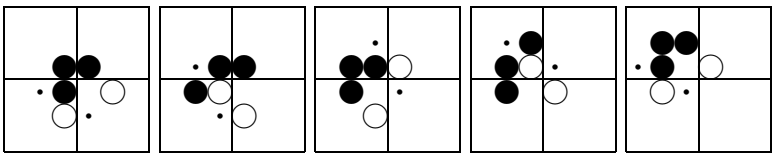
```
// bepaal som van array-elementen uit eerste rijen van A
int somarray (int A[ ][10], int rijen) {
  int i, j, som = 0;
  for (i = 0; i < rijen; i++)
    for (j = 0; j < 10; j++)
```

```
      som += A[i][j];
  return som;
}  // somarray
```

De value-parameter `rijen` geeft het aantal rijen door, maar het aantal kolommen moet een `const` zijn, bijvoorbeeld 10. Dat moet de compiler weten om bij bijvoorbeeld `A[3][5]` het betreffende adres te kunnen berekenen. Dat adres is hier `A+3*10+5`, of eigenlijk `A+(3*10+5)*sizeof (int)`, want bij `+` wordt met de stapgrootte, dat wil zeggen de grootte van de array-elementen in bytes, rekening gehouden. De waarde van `A` is hier het adres waar het array begint. De rijen van het array liggen vanaf dit adres achter elkaar in het geheugen, en de rijlengte (het aantal kolommen) moet dus van te voren vastliggen. De ontwerpers van C++ hebben uit efficiency-overwegingen voor deze aanpak gekozen. De "unaire" operator `sizeof` geeft de grootte van het betreffende type—of de variabele—in bytes.

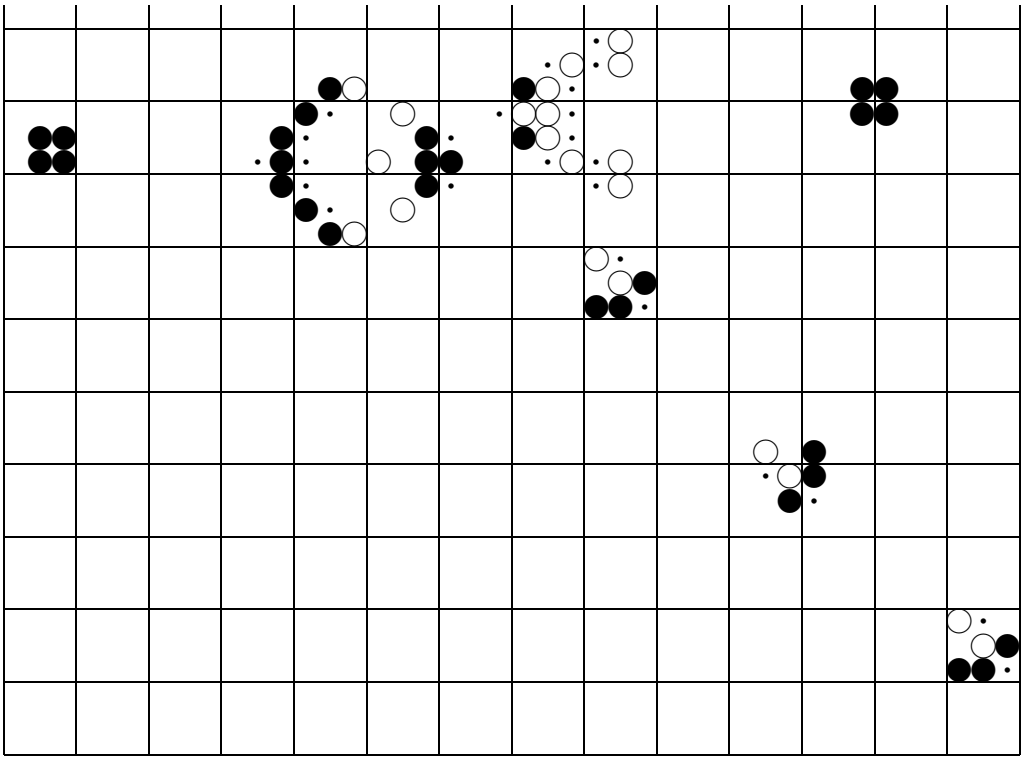
3.7 Wat is Life?

Life is een spel, maar wel een heel bijzonder spel. Het is geen een- of tweepersoonsspel, maar een nulpersoonsspel. Er zijn spelregels die het spel bepalen, maar het enige wat je zelf mag doen is het bepalen van de begintoestand van het speelbord. Je stelt als het ware wat getallen in en vervolgens leun je achterover in je stoel en je kijkt wat de computer voor mooie plaatjes te voorschijn tovert. Bij Life gaat het om een kolonie levende cellen, waarbij er sommige dood gaan, terwijl op andere plekken leven ontstaat. Zo zie je grillige, maar soms ook zeer regelmatige patronen ontstaan.
Het spel Life is omstreeks 1970 bedacht door de Engelse wiskundige J.H. Conway. De regels voor Life zijn erg simpel. Life wordt gespeeld op een oneindig groot bord met vierkante vakjes, de zogenaamde cellen. Cellen zijn dood of levend. Elke cel heeft precies acht buren (diagonaal doet ook mee). Een dode cel wordt in de volgende generatie levend als zij precies drie levende buren had. Een levende cel blijft leven als zij precies twee of drie levende buren had, anders gaat zij dood (hetzij door eenzaamheid, hetzij door overbevolking).
Het spelen van het spel is nu het geven van een zogenaamde beginconfiguratie, dat wil zeggen een eindig aantal levende cellen, en vervolgens het kijken naar de ontwikkeling die deze groep cellen in de tijd ondergaat. Een klein voorbeeld staat hieronder. Hierbij gaan de open rondjes dood, en de zwarte rondjes blijven leven; op de plek van de stippen ontstaat nieuw leven. Het stelletje cellen dat je hier ziet wordt wel de *glider* genoemd; als je het van een afstandje ziet lijkt het inderdaad net of de kolonie zich glijdend verplaatst.



Wat is er nu interessant aan dit spel, afgezien van de soms mooie plaatjes die het oplevert? Er blijken zeer ingewikkelde vragen aan vast te zitten, die binnen de informatica vaak terugkomen. Is het bijvoorbeeld mogelijk om bij een gegeven beginconfiguratie te voorspellen of de kolonie al of niet uitsterft, of misschien zelfs willekeurig groot wordt? Dat deze vraag niet zo simpel is als het lijkt blijkt wel door eens te kijken wat er gebeurt

wanneer je als beginconfiguratie een aantal cellen op een rechte lijn, naast elkaar dus, neemt. Bij 1 of 2 levende cellen naast elkaar sterft de boel meteen uit, bij 3 of 5 naast elkaar krijg je fraai knipperende kolonies, bij 4 naast elkaar komt de zaak na enige tijd tot stilstand. Een regelmaat is nog niet te ontdekken, en dat wordt nog erger als je als beginconfiguratie naar nog meer levende cellen naast elkaar kijkt; 15 naast elkaar sterven uit, maar 16 naast elkaar geven een flink aantal knipperende cellen.
In 1970 won een groepje onderzoekers van het beroemde MIT in Boston een prijs van "slechts" $ 50 door een beginconfiguratie te fabriceren waarbij het aantal levende cellen groter en groter wordt. Ze noemden het de *glider gun*, omdat hij elke dertigste generatie een nieuwe glider als het ware afvuurt (zie het plaatje). Het bepalen of een willekeurige beginconfiguratie op den duur al of niet volledig uitsterft is ongelooflijk moeilijk: het beroemde, enige jaren geleden door Wiles opgeloste probleem van Fermat (er zijn geen gehele getallen $a > 0$, $b > 0$, $c > 0$ en $n > 2$ met $a^n + b^n = c^n$) is een speciaal geval ...



Een ander aspect van Life is dat je er zelfs computers mee kan bouwen, of liever gezegd mee kan nabootsen. Zo is het mogelijk, maar wel uiterst ingewikkeld, om NOT-, AND- en OR-poorten te "maken" door geschikte combinaties van gliders en guns op heel speciale plekken ten opzichte van elkaar te plaatsen. Op deze manier kun je laten zien dat vragen over computers eigenlijk vragen over Life zijn, en omgekeerd!
Er is heel veel over Life geschreven, onder andere door Martin Gardner in de Scientific American. Een goed, maar moeilijk, boek waarin veel is te vinden over Life (en trouwens ook over allerlei andere spellen zoals Nim en de kubus van Rubik) is "Winning ways for your mathematical plays" van E.R. Berlekamp, J.H. Conway en R.K. Guy. Via World Wide Web (WWW) is er ook veel moois te bekijken op Life-gebied, kijk maar eens naar

`http://radicaleye.com/lifepage/`

Het programmeren van een spel als Life is in eerste instantie niet al te moeilijk. In een taal als C++ kun je in een dag zo'n programma in elkaar zetten. Een probleem hierbij is wel dat je niet of nauwelijks een oneindig groot "speelbord" kunt maken in de computer; soms neem je genoegen met alleen de grootte van een beeldscherm, waarbij cellen niet mee doen als ze van het scherm afvallen, of waarbij je ze bijvoorbeeld bovenin het scherm laat binnenkomen als ze er beneden afvallen. Je maakt dan wel een ander spel, wat je al aan een glider gun kunt zien: soms past deze niet eens op een scherm! Een ander punt is dat een en ander op het scherm soms zeer traag kan verlopen; als dat gebeurt wil het wel eens zinnig zijn om het programma, of stukken ervan, in Assembler te schrijven. Dat zit heel dicht tegen de processor van de computer aan en je kunt dan heel uitgekiende snelle programma's maken, waarbij je helaas wel de kracht van een gestructureerde taal als C++ mist. Er zijn overigens heel goede en snelle gratis verkrijgbare Life-programma's voor de PC, zie bijvoorbeeld bovenstaand web-adres.
Waarschijnlijk komt al snel een array `bool Life[MAX][MAX];` in beeld, om daarmee de populatie te beschrijven: als `Life[i][j]` de waarde `true` heeft, is in de `i`-de rij, `j`-de kolom leven. Door de `const int MAX` groot genoeg te kiezen kan er nog een redelijk resultaat bereikt worden. De volgende generatie wordt steeds op eenvoudige wijze uit de huidige verkregen,

3.8 Sorteren en zoeken

Van oudsher hebben algoritmes voor sorteren en zoeken in de belangstelling gestaan. We zullen er hier enkele kort behandelen. Steeds gebruiken we een array van het volgende type:

```
// Arraygrootte:
const int n = 20;
// Een array A:
int A[n];
```

Eigenlijk moeten we de array-grootte `n` als parameter doorgeven. Denk eraan dat arrays altijd bij `0` beginnen te nummeren, en doorlopen tot en met `n-1`. Let er op de array-grenzen niet te overschrijden! Laten we als voorbeeld eens het kleinste getal uit een array opsporen:

```
// Geef kleinste getal uit array A (dat n elementen heeft)
int minimum (const int A[ ], int n) {
  int klein = A[0];
  int i;
  for (i = 1; i < n; i++)
    if ( A[i] < klein )  // kleinere gevonden
      klein = A[i];
  return klein;
}  // minimum
```

Arrays en pointers hebben in C++ een nauwe band. Wij zullen proberen de twee zaken niet te verwarren, en ons niet bezighouden met zaken als de gelijkheid van `&A[0]` en `A` voor een variabele `A` (`int A[n]`). Zo mag bijvoorbeeld in de heading van de vorige functie ook `const int * A` staan. Eigenlijk staat hier voor de compiler de volgende informatie:

op deze plek staat een serie `int`'s; hoeveel dat er zijn is "niet van belang". De tweede parameter, `n`, wordt gebruikt om dit aantal door te geven. De `const` zorgt er hier voor dat je de array-elementen niet mag wijzigen. In bovenstaande functie zou `A[0] = klein;` een waarschuwing van de compiler opleveren.

3.8.1 Lineair zoeken

We zoeken in een (ongesorteerd) array naar een getal. Voor de hand ligt het volgende. Vooraan (of achteraan) beginnen en element voor element vergelijken tot we het gezochte element gevonden hebben of tot alle elementen geweest zijn, en het getal klaarblijkelijk niet voorkomt.

```
// Zoek getal in array A met n elementen. Lineair zoeken.
// Geeft index met A[index] = getal, als getal tenminste
// voorkomt; zo niet: resultaat wordt -1.
int lineairzoeken (int A[ ], int n, int getal) {
  int index = 0;
  bool gevonden = false;
  while ( !gevonden && ( index < n ) ) {
    if ( getal == A[index] )
      gevonden = true;  // of meteen return index;
    else
      index++;
  }  // while
  if ( gevonden )
    return index;
  else
    return -1;
}  // lineairzoeken
```

Als je pech hebt, bijvoorbeeld als het gezochte getal niet voorkomt, kost je dat voor een array met n elementen n vergelijkingen; kortom: $O(n)$ (lineaire orde); zie de opgaven. De methode heet *lineair zoeken*.

3.8.2 Binair zoeken

Als het array gesorteerd is kunnen we wat slimmer zijn. In een telefoonboek bijvoorbeeld zal iedereen die een naam zoekt (om het bijbehorende telefoonnummer te krijgen) gebruik maken van de alfabetische ordening. Zoek je overigens naar een nummer (om te weten wie dat nummer heeft), dan zit er—tenzij je bij de telefoondienst werkt—niets anders op dan met lineair zoeken het hele telefoonboek door te nemen.
Binair zoeken is een voor de hand liggende methode om een getal op te sporen in een gesorteerd array. Kijk allereerst of het middelste element het gezochte element is. Zo niet, bepaal dan op grond van vergelijken met dat middelste element of het zoekproces voortgezet moet worden in de linker helft of juist in de rechter helft van het array en herhaal dit. Stop zodra het element gevonden is, danwel het te onderzoeken array leeg is. Als het aantal elementen even is: kies één van de twee middelste.

```
// Zoek getal in GESORTEERD array A met n elementen. Binair zoeken.
```

```
// Geeft index met A[index] = getal, als getal tenminste
// voorkomt; zo niet: resultaat wordt -1.
int binairzoeken (int A[ ], int n, int getal) {
  int links = 0, int rechts = n-1;  // zoek tussen links en rechts
  int midden;
  bool gevonden = false;
  while ( !gevonden && ( links <= rechts ) ) {
    midden = ( links + rechts ) / 2;
    if ( getal == A[midden] )
      gevonden = true;  // of meteen return midden;
    else
      if ( getal > A[midden] )
        links = midden + 1;
      else
        rechts = midden - 1;
  }  // while
  if ( gevonden )
    return midden;
  else
    return -1;
}  // binairzoeken
```

Een aanroep ziet eruit als `index = binairzoeken (A,n,getal)`. In het slechtste geval ben je hier, als er bijvoorbeeld $n = 2^k - 1$ (met $k > 0$ geheel) elementen zijn, k vergelijkingen aan kwijt; men zegt wel: $O(\log n)$ (logaritmische orde).

3.8.3 Een eenvoudige sorteermethode

Nu willen we een array (rij) op grootte oplopend sorteren. Een heel eenvoudige methode is de volgende. Beginsituatie: het gesorteerde stuk is leeg en het ongesorteerde stuk is de hele rij. Zoek nu eerst het kleinste element in het ongesorteerde stuk en verwissel dat met het voorste element van dat stuk. Het gesorteerde stuk is nu één element groter en het ongesorteerde stuk is één element kleiner. Herhaal dit totdat het ongesorteerde stuk leeg is. Dit wordt in C++:

```
void simpelsort (int inhoud[ ], int n) {
  int voorste, kleinste, plaatskleinste, k;
  for (voorste = 0; voorste < n; voorste++) {
    plaatskleinste = voorste;
    kleinste = inhoud[voorste];
    for (k = voorste + 1; k < n; k++)
      if ( inhoud[k] < kleinste ) {
        kleinste = inhoud[k];
        plaatskleinste = k;
      }  // if
    if ( plaatskleinste > voorste )
      wissel (inhoud[plaatskleinste],inhoud[voorste]);
  }  // for
}  // simpelsort
```

Nogmaals: de eerste parameter is een array. De *actuele* lengte van dit array wordt als tweede parameter meegegeven; zelfs als de eerste parameter `int inhoud[n]` was geweest—waar de C++-compiler overigens niets anders mee doet—moeten we nog steeds goed op de array-grenzen letten!

3.8.4 Bubblesort

Een eenvoudige variant hierop is de methode *bubblesort*. De C++-code is bijzonder compact, maar bubblesort is helaas niet zo'n goede sorteermethode. Het sorteren van een rijtje met n getallen kost altijd $\frac{1}{2}n(n-1)$ vergelijkingen; vaak zegt men: $O(n^2)$ (kwadratische orde). Meer ingewikkelde methodes als Shellsort en quicksort doen dat soms of vaak (mischien zelfs altijd) sneller.
Een array, `A` geheten, wordt als volgt op grootte gesorteerd. In de eerste ronde worden `A[0]` en `A[1]` vergeleken en indien nodig (namelijk als `A[0]` groter is dan `A[1]`) wordt hun inhoud verwisseld; daarna `A[1]` en `A[2]`, ..., `A[n-2]` en `A[n-1]`; het is duidelijk dat het grootste element nu achteraan staat. In de tweede ronde worden `A[0]` en `A[1]` vergeleken, ..., `A[n-3]` en `A[n-2]`; er vindt dus één vergelijking minder plaats. Zo gaat dit verder; in de laatste (de (`n-1`)ste) ronde hoeven alleen nog maar `A[0]` en `A[1]` vergeleken te worden. De grote getallen "borrelen" als het ware naar achteren, vandaar de naam bubblesort.

```
void bubblesort (int A[ ], int n) {
  int ronde, j;
  for (ronde = 1; ronde < n; ronde++)
    for (j = 0; j < n - ronde; j++)
      if ( A[j] > A[j+1] )
        wissel (A[j],A[j+1]);
}  // bubblesort
```

3.8.5 Invoegsorteer

Als een rij reeds gesorteerd is en men wil een element toevoegen, dan kan *invoegsorteer*, oftewel *insertion sort*, gebruikt worden: zoek de plaats op waar het element moet komen en voeg het element op die plaats ertussen (bij een array betekent dat, dat het achterstuk één plaats naar achter moet schuiven).
Een ongesorteerde rij kan men met invoegsorteer sorteren door element voor element aan een oorspronkelijke lege rij (die is namelijk "per definitie" gesorteerd) toe te voegen zoals beschreven. De complexiteit is vergelijkbaar met die van bubblesort. De C++-code laten we als opgave: Opgave 45.

4 Opgaven

De uitwerkingen van de opgaven zijn via WWW te vinden, zie

`http://www.liacs.nl/home/kosters/lst/`

1. Geef type en zo mogelijk de waarde van de volgende uitdrukkingen. Hierbij zijn `p`, `q`, `r` en `s` variabelen van type `bool`, en `k` van type `int`.

```
a. sqrt (4)          b. sqrt (4.0)         c. 't' - 'e'
d. ceil (-99.9)      e. - floor (99.9)     f. - floor (-99.9)
g. !(p && q) == !(!p && !q)
h. 10 % 3            i. 10 / 3             j. 126 / 3 % 5
k. ( p && (q && !q) ) || !( r || (s || !s) )
l. ( floor (-65.3) < fabs (-65.3) ) && p
m. odd (k) || odd (k+1)
```

2. a. Schrijf een C++-programma dat de gebruiker om een temperatuur in graden Fahrenheit als invoer vraagt. Na het inlezen hiervan wordt deze temperatuur omgerekend in graden Celsius, waarna beide getallen afgedrukt worden. Gebruik: Temp(in Celsius) = (5/9) $*$ (Temp(in Fahrenheit) – 32).

   b. Idem, maar nu mag alleen met gehele getallen gerekend worden. In dit geval moet er dus afgerond worden.

3. Schrijf een C++-programma dat het aantal seconden vanaf middernacht aan de gebruiker vraagt, en vervolgens de tijd uitvoert als: 'uren:minuten:seconden'.

4. Wat is het grootste gehele getal dat in C++ in een `int` past? Waarom dit getal? Hangt dit af van de C++-implementatie? En hoe staat het bij een `long`? Wat gebeurt er als je `1` optelt bij het grootste te representeren getal?
   Wat is het grootste getal dat in een `double` past? En het kleinste positieve (groter dan nul) getal?

5. Geef kritiek op de het volgende stukje C++, waarbij de variabele `doorgaan` van het type `bool` is: `if ( doorgaan = true ) ...`

6. a. Laat zien dat elk do-while-statement herschreven kan worden met behulp van if- en while-statements.

   b. Laat zien dat elk while-statement herschreven kan worden met behulp van if- en do-while-statements. Kan het ook zonder if-statements?

7. Een slecht geschreven C++-programma bevatte het volgende statement:

```
if ( a < b ) if ( c < d ) x = 1;
else if ( a < c ) if ( b < d ) x = 2;
                  else x = 3;
else if ( a < d ) if ( b < c ) x = 4;
                  else x = 5;
                  else x = 6;
                  else x = 7;
```

a. Herschrijf dit statement; gebruik een betere layout.

b. Staan er overbodige of tegenstrijdige condities in?

c. Schrijf een eenvoudiger statement dat hetzelfde effect heeft.

8. Schrijf een programma dat drie gehele getallen inleest en deze in volgorde van grootte afdrukt. Geef enkele verschillende methoden. Zo kunnen de getallen alleen op het beeldscherm gesorteerd afgedrukt worden, maar ook intern in het programma gesorteerd worden.

9. Schrijf een programma dat een geheel getal n inleest, en vervolgens (als tenminste $n \geq 0$) $n! = n * (n-1) * \cdots * 2 * 1$ berekent. Als $n < 0$ moet er een passende foutmelding afgeleverd worden. Gebruik hierbij:

   a. het while-statement,

   b. het for-statement,

   c. het do-while-statement.

   Welke mogelijkheid levert het "beste" programma?

   Schrijf ook een functie (zonder recursie) die $n!$ berekent.

10. Schrijf een functie `void spaties (int aantal)` die `aantal` spaties op het beeldscherm zet.

11. Schrijf een programma dat als invoer om een rij positieve getallen vraagt, afgesloten door een getal kleiner dan of gelijk aan nul. Vervolgens moeten het maximum, het minimum en het gemiddelde van deze getallen (het laatste getal niet inbegrepen) bepaald worden. Is het for-statement of het while-statement beter geschikt?

12. Schrijf een programma dat voor elk bedrag kleiner dan een euro het kleinste aantal munten bepaalt dat samen dit bedrag oplevert. Gebruikt mogen worden: munten van één, twee, vijf, tien, twintig en vijftig eurocent. De gebruikte munten moeten ook worden afgedrukt.

13. Schrijf een programma dat uitdrukkingen als `+56-664-77+2-888;` inleest en berekent. Elk (geheel) getal wordt voorafgegaan door een teken, terwijl de hele uitdrukking door een punt-komma wordt afgesloten. Schrijf ook een versie die echt karakter voor karakter inleest (met `kar = cin.get( );`).

14. Schrijf een programma dat voor gegeven gehele n het volgende berekent:

$$A(n) = 1/2 + 2/4 + 3/8 + 4/16 + \cdots + n/2^n.$$

    Gebruik een functie; maak hierbij *geen* aparte functie voor machtsverheffen. Maakt het overigens uit in welke volgorde de sommatie berekend wordt?

15. Voorspel de uitvoer van de volgende twee programma's:

```
#include <iostream>
using namespace std;
int iets;
```

```
#include <iostream>
using namespace std;
int iets;
```

```
void wat (int& a, int& b) {
  a = -1;
  b = -2 * a;
  cout << a << b << endl;
}  // wat
int main ( ) {
  iets = 1;
  wat (iets,iets);
  cout << iets << endl;
  return 0;
}  // main
```

```
void bla (int &a, int & b) {
  a = 10 * b + iets / 2;
  cout << a << b << endl;
}  // bla

int main ( ) {
  iets = 10;
  bla (iets,iets);
  cout << iets << endl;
  return 0;
}  // main
```

Wat gebeurt er als `&` weggelaten wordt in de headings?

16. Gegeven is de functie:

```
void test (int x, int& y) {
  int z = 9;
  x = 5;
  y = 6;
  z = 7;
}  // test
```

Stel dat `x` 1 is, `y` 2 en `z` 3 voor globale variabelen `x`, `y` en `z` (van type `int`). Wat is de uitvoer geproduceerd door:

a. `test (x,y); cout << x << y << z << endl;`

b. `test (y,x); cout << x << y << z << endl;`

c. `test (1,z); cout << x << y << z << endl;`

d. `test (z,1); cout << x << y << z << endl;`

e. `test (z,x); cout << x << y << z << endl;`

17. Gegeven een functie `F`:

```
int F (int& x) {
  x = x + 4;
  return x;
}  // F
```

a. Wat gebeurt er bij `cout << (x + F(x));` (`x` is een globale variabele van type `int` met waarde 7)?

b. Wat gebeurt er bij `cout << (F(x) + x);` ?

c. Idem, als `&` weggelaten wordt.

d. Is het verstandig als functies die een `int` als resultaat hebben van "call by reference" gebruik maken?

18. Gegeven zijn de volgende functies:

```
int f (int x, int y) {
  x--; return x*y; }  // f
int g (int a, int b) {
  int x = 3; b += x; a--; a = f (a,b) + f (a,a);
  cout << x << a << b; return a+x-2; }  // g
```

a. Neem aan dat de waardes van de *globale variabelen* `x` en `y`, beide `int`, bij binnenkomst van `g` 6 respectievelijk 16 zijn.

Wat gebeurt er bij `cout << g (x,y) << x << y << endl;`? Wat wordt er afgedrukt? Probeer duidelijke uitleg te geven.

b. Geef een functie `int G (int a, int b)` die dezelfde return-waarde oplevert als `g` voor alle mogelijke waardes van de parameters, maar die uit slechts één `return`-statement bestaat, en waarin `f` niet meer wordt aangeroepen.

c. We voegen vier maal een `&` toe, en wel bij alle parameters in de headings van `f` en `g`. Beantwoord opnieuw vraag a. Geef alle mogelijke uitvoeren, en leg uit waarom er verschillende antwoorden mogelijk zijn.

19. Gegeven zijn de volgende functies:

```
int peter (int r, int s) {
  s--; return r+s+2; }  // peter
int ellen (int p, int q) {
  int a = 7; p++; q -= 2;
  for ( a = 2; a < q; a++ ) p = p + peter (p,q);
  cout << a << p << q << endl; return a+p+q; }  // ellen
```

a. Neem aan dat de waardes van de *globale variabelen* `a` en `b`, beide `int`, bij binnenkomst van de functie `ellen` `2` respectievelijk `6` zijn.
Wat gebeurt er bij `cout << ellen (a,b); cout << a << b << endl;`?
Wat wordt er afgedrukt? Probeer duidelijke uitleg te geven.

b. We voegen vier maal een `&` toe, en wel bij alle parameters in de headings van `ellen` en `peter`. Beantwoord opnieuw de vorige vraag.

c. Vervang in de functie `ellen` het statement `p = p + peter (p,q);` door het statement `p = p + peter (q,p);`. Zet de vier `&`'s er weer bij, net als bij b. Leg uit waarom uiteindelijk de globale variabele `a` verschillende waarden kan hebben, en geef deze.

20. Schrijf een functie die `x` tot de macht `y` berekent, waarbij

a. `x` en `y` beide van type `double` zijn,

b. `x` van type `double` is, `y` van type `int`, en `exp` en `log` (uit `math.h`) niet gebruikt mogen worden.

Gebruik "overloading". Vergelijk het resultaat eens met `pow` uit `math.h`.

21. Schrijf een functie `drukaf (char letter)` die de letters a, b en c afdrukt als:

```
****        ****        ****
*  *        *  *        *
****   ,    ****   ,    *      .
*  *        *  *        *
*  *        ****        ****
```

22. Schrijf een functie `Pyramide (int hoogte)` die een pyramide, bestaande uit sterretjes, en ter hoogte `hoogte` afdrukt (zie beneden voor `hoogte` = 4). Er moet getest worden of `hoogte` positief is. Als `hoogte` negatief is gebeurt er niets.

```
   *
  ***
 *****
*******
```

23. Schrijf een functie die elk voorkomen van het woordje `"er"` in een (onzin)tekst verandert in `"ER"`; tevens moet het aantal regels van de tekst bepaald worden.

24. Schrijf een functie die voor elke ingevoerde datum tussen 1 januari 1900 en 31 december 2100 de bijbehorende dag afdrukt. Gegeven is dat 1 januari 2002 op een dinsdag viel. Het jaar 2000 is overigens wel een schrikkeljaar, maar 1900 en 2100 niet.

25. Schrijf een functie `fibo (int n)` die het `n`-de getal van Fibonacci `Fibo(n)` berekent. Er geldt `Fibo(1)` = `Fibo(2)` = 1 en `Fibo(n) = Fibo(n-1) + Fibo(n-2)`, voor `n` > 2 (gebruik nog geen recursie, zie later).

26. Schrijf een functie die waarden van de zogenaamde Hermite-polynomen uitrekent (zie ook later voor een recursieve versie). Er geldt: $H_0(x) = 1$, $H_1(x) = 2x$ en $H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x)$ voor $n > 1$.

27. In een zeker C++-programma moeten de functies `P`, `Q` en `R` voorkomen. In de functie `P` wordt gebruik gemaakt van de functies `Q` en `R`. Geef zoveel mogelijk verschillende manieren om dit te programmeren.

28. Het te betalen bedrag voor het sturen van een pakketje overzee wordt als volgt berekend (nemen we aan). Eerst wordt het gewicht naar boven afgerond op het dichtstbijzijnde veelvoud van 15 gram. Dan wordt het tarief uit de volgende tabel afgelezen:

    - Als het gewicht 15 gram is, wordt het tarief 120 eurocent,
    - Als het gewicht 30 gram is, wordt het tarief 220 eurocent,
    - Als het gewicht 45 gram is, wordt het tarief 310 eurocent,
    - Als het gewicht 60 gram is, wordt het tarief 360 eurocent, plus 20 eurocent per volledige 1000 km,
    - Als het gewicht 75 gram of meer is, wordt het tarief 400 eurocent, plus 30 eurocent per volledige 1000 km.

a. Maak een switch-statement dat bij gegeven gewicht en—indien nodig—de afstand het tarief uitrekent. Maak hier ook een functie van.

b. De invoer bestaat nu uit een aantal regels. Elke regel bevat de gegevens van een te versturen pakketje: het gewicht (in grammen) en daarna de af te leggen afstand (in kilometers). De laatste regel bevat alleen het getal 0. Schrijf een programma dat de verzendkosten van al de pakketjes samen berekent. Bovendien moet worden bijgehouden hoeveel pakketjes uit elke gewichtscategorie verstuurd worden.

29. Schrijf een programma dat een `int array[100]` vult met de eerste honderd kwadraten. Maak hier ook een functie van. Druk het array tevens af.

30. Er is een enquete gehouden onder honderd mensen. De enquete bestond uit drie vragen. Op elke vraag waren slechts de antwoorden ja en nee mogelijk. De resultaten staan in een array `bool uitslag[100][3]`. Zo bevat `uitslag[30][1]` het antwoord van persoon 31 op vraag 2, waarbij `true` ja betekent en `false` nee. Schrijf een functie die voor elk van de acht mogelijke antwoord-combinaties bepaalt bij hoeveel mensen deze voorkomt.

31. Gegeven zijn:

```
const int m = 30; const int n = 40;
char puzzel[m][n]; int nummers[m][n];
```

In een array `puzzel` zit een kruiswoordraadsel opgeslagen. De zwarte vakjes worden met `'#'` aangegeven, woorden staan in hoofdletters. Een voorbeeld ($\mathtt{m} = 3$, $\mathtt{n} = 4$):

```
H E T #         1 0 2 0
A # O M         0 0 3 4
# S P A         0 5 0 0
```

a. Schrijf een boolese C++-functie `BestaatHoriWoord (puzzel, i, j)` die precies dan `true` oplevert als op positie (`i`, `j`), dus beginnend in de `i`-de rij en de `j`-de kolom, een horizontaal woord begint. Het woord moet meer dan één letter hebben, en echt op deze plek beginnen. Aangenomen mag worden dat (`i`, `j`) binnen de puzzel valt.

b. Schrijf een C++-functie `Nummeren (puzzel, nummers)` die de vakjes van `nummers` nummert zoals de puzzel uit `puzzel` aangeeft (zie voorbeeld). Vakjes waar geen woord begint, bijvoorbeeld de met een `'#'` gemarkeerde, krijgen een 0. Vakjes waar wel een horizontaal en/of verticaal woord begint, krijgen regel voor regel, van links naar rechts, een rangnummer, beginnend bij 1. Gebruik de functie van a, en een soortgelijke functie voor verticale woorden (deze hoeft niet meer geschreven te worden).

c. Schrijf een C++-functie `Woord (puzzel, w)` die het `w`-de woord uit puzzel `puzzel` afdrukt, als hier een horizontaal woord staat, en anders niets. Neem aan dat de puzzel minimaal `w` woorden heeft. In het voorbeeld wordt bij 1 `HET` afgedrukt, bij 2 niets en bij 3 `OM`.

32. In een array `T` (`int T[m][n]` met `m` en `n` constanten) wordt van `m` personen bijgehouden wie op welke tijdschriften geabonneerd is. Zo betekent `T[3][2] == 13` dat klant `3` tijdschrift `13` leest; de `2` heeft geen speciale betekenis. Als een array-element echter `0` is, wordt het nog niet, of niet meer, gebruikt voor een tijdschrift—de persoon in kwestie heeft dan dus minder dan `n` abonnementen, het maximale aantal. Per klant zijn de tijdschriftnummers verschillend en oplopend gesorteerd, maar er kunnen nullen tussen de niet-nullen zitten.
Twee voorbeelden die dezelfde situatie beschrijven (met `m == 3` en `n == 4`):

```
0  3 10 13          3 10 13  0
0  0 10  0         10  0  0  0
1  5  0 10          1  5 10  0
```

a. Schrijf een `int` C++-functie die de persoon met de meeste abonnementen (de index van de rij met de meeste niet-nullen) geeft. Als er meer dan één persoon is die het grootste aantal abonnementen heeft, doet het er niet toe welke wordt opgeleverd. In het voorbeeld hierboven: `0` of `2`.

b. Schrijf een `void` C++-functie die in iedere rij de niet-nullen zoveel mogelijk naar links aanschuift—van het voorbeeld linksboven naar dat rechtsboven.

c. Schrijf een `int` C++-functie die oplevert hoeveel tijdschriften door meer dan één persoon gelezen worden. In het voorbeeld zou het antwoord `1` zijn. Als het helpt, mag aangenomen worden dat de tijdschriftnummers tussen `1` en `999` liggen, grenzen inbegrepen.

33. Schrijf een eenvoudig Life-programma.

34. Geef een programma dat een tekst regel voor regel inleest en elke regel van achter naar voren afdrukt. Hierbij moet een `char regel[80]` gebruikt worden om de regel in op te slaan, aannemend dat er hoogstens 80 (of 79) tekens per regel staan. Wat verandert er als `char*` gebruikt moet worden?

35. Schrijf een programma dat een tekst letter voor letter leest en het langste woord dat voorkomt in een afdrukt.

36. Schrijf een programma dat alle in een invoertekst voorkomende woorden in een array opslaat. Als woorden twee keer (of nog vaker) voorkomen moeten ze maar één keer opgeslagen worden.

37. Schrijf een functie `vermenigvuldig (int A[ ][n], int v[ ], int res[ ])` die het product van de matrix `A` en de vector `v` in de vector `res` stopt. Neem aan dat `A` een `n` bij `n` matrix is en `v` een vector met `n` componenten.

38. a. Bedenk een array-representatie voor een stand op een schaakbord.

b. Is er een betere (minder geheugenruimte) denkbaar?

c. Idem voor een stand op een dambord.

39. a. Schrijf een functie die de array-indices oplevert van het grootste en van het op een na grootste getal van een gegeven array `int A[max]`, met `max` $\geq$ `2`.

b. Geef in woorden een methode aan die ditzelfde bewerkstelligt, maar dan met minder vergelijkingen tussen gehele getallen. Hint: hoe vind je de twee echt beste spelers van een tennistoernooi?

40. Gegeven zijn de volgende declaraties in C++:

```
char woord[m];
char verhaal[n];
```

met `m` en `n` constanten (gehele getallen) met `n` $>$ `m` $>$ 0. Schrijf nu een C++-functie die vertelt of een variabele `woord` voorkomt in een variabele `verhaal`. Als dat zo is moet de array-index die het eerste optreden van `woord` aanduidt afgedrukt worden.

Enkele voorbeelden (in string-notatie):

- `woord = "la"` (`m = 2`), terwijl `verhaal = "leuter babbel bla bla"` (met `n = 21`), met als uitvoer: `Ja, index = 15`.
- `woord = "tja"` (`m = 3`), `verhaal` als boven, levert als uitvoer: `Nee`.

41. Gegeven een array `bord`: `char bord[8][8];`. Dit is dus een "schaakbord" gevuld met letters.

a. Schrijf een boolese functie in C++ die `true` oplevert als er in `bord` minstens twee aangrenzende (naast elkaar of boven elkaar gelegen) vakjes zijn met dezelfde letter, en anders `false`.

b. Idem, maar nu moet er `true` opgeleverd worden als er alleen hoofdletters in het array zitten opgeslagen.

c. Neem aan dat er alleen hoofdletters worden opgeslagen. Schrijf een functie die bepaalt of er tenminste een rij in `bord` is die geheel uit klinkers (`'A'`, `'E'`, `'I'`, `'O'`, `'U'`) bestaat. De functiewaarde moet het nummer van zo'n rij zijn en `-1` als zo'n rij niet bestaat.

d. Herschrijf de functie van c zo dat er geen overbodige tests uitgevoerd worden. Bijvoorbeeld: als er een rij met alleen klinkers gevonden is hoeven de overige rijen niet meer bekeken te worden.

42. a. Bedenk zelf methoden om willekeurige ("random") getallen te genereren. Geef voor- en nadelen.

b. Bekijk de lineaire congruentie methode. Bereken `y = ( 3 * x + 5 ) % 16;` , waarbij `x` om te beginnen `1` is, dus `y` wordt `8`: het eerste random-getal. Vervang vervolgens `x` door `y` en bereken de nieuwe `y`; deze is `13`: het tweede random-getal. Ga zo door. Reken de rij verder uit. Wat zijn (lijken) voor- en nadelen van deze methode?

c. Probeer de getallen van b zo te kiezen dat de methode beter loopt.

43. Lineair zoeken is een voor de hand liggende methode om een getal in een array op te zoeken. In een reeds gesorteerd array kan een getal snel worden opgespoord met binair zoeken.

a. Gegeven de rij 1, 3, 9, 10, 13, 17, 19, 21, 28. Hoe verloopt het zoeken naar het getal 9? En naar 21? En naar 18? Beantwoord de vraag zowel voor lineair als voor binair zoeken.

b. Stel dat het aantal elementen in de rij $2^k - 1$ (met $k > 0$ geheel) is, bijvoorbeeld $8 - 1 = 7$ of $16 - 1 = 15$. Hoeveel vergelijkingen heb je nodig om het eerste array-element te vinden? En hoeveel om te constateren dat een bepaald element niet voorkomt? Beantwoord de vraag wederom zowel voor lineair als voor binair zoeken.

44. a. Pas de sorteermethode bubblesort zo aan dat er gestopt wordt zodra er een ronde zonder verwisselingen geweest is. Pas de methode vervolgens zo aan dat bij iedere ronde begonnen wordt bij het eerste element dat wellicht verkeerd zou kunnen staan, en geëindigd wordt bij het laatste element dat wellicht verkeerd zou kunnen staan (op grond van kennis uit de vorige ronde).

    b. Hoeveel vergelijkingen van array-elementen doet de oorspronkelijke bubblesort, respectievelijk de aangepaste versie, bij het sorteren van n getallen in het beste geval en in het slechtste geval?

45. Geef C++-code voor invoegsorteer.

46. Gegeven zijn `const int n = 1000;` en `int A[n];`. De variabele `A` bevat een rij onderling verschillende getallen.

    a. Schrijf nu een C++-functie `bergop (A,i,n)` die het array-element `A[i]` (met `i` tussen 0 en `n`-2) op de juiste plaats opbergt tussen de reeds oplopend gesorteerde array-elementen `A[i+1]`,`A[i+2]`,...,`A[n-1]`.

    b. Schrijf een C++-functie `sorteer (A)` die het array `A` oplopend sorteert door de functie `bergop` herhaald aan te roepen.

    c. Hoeveel vergelijkingen tussen array-elementen doet dit sorteer-algoritme voor het rijtje n, $n - 1$,...,1?

    d. Is de methode beter dan, gelijkwaardig met of slechter dan de gewone bubblesort qua complexiteit (gedaan aantal vergelijkingen), zowel in het beste als in het slechtste geval?

5 Oude tentamens

Er volgen drie oude tentamens met uitwerkingen. Op WWW staat nog meer!

5.1 Tentamen woensdag 3 april 2002

De opgaven tellen alle vier even zwaar mee. Veel succes!

1. **a.** Schrijf een C++-functie `int max2 (int x, int y)` die het grootste van de twee getallen `x` en `y` teruggeeft (met behulp van een `return`-statement).
b. Schrijf een C++-functie `int max4 (int x, int y, int u, int v)` die het grootste van de vier getallen `x`, `y`, `u` en `v` teruggeeft. De functie moet gebruik maken van de functie `max2` van **a**; het kan in één regel.
c. Schrijf een C++-functie `void som (int n)` die de som $1\times2+2\times3+3\times4+\cdots+(\mathtt{n}-1)\times\mathtt{n}$ uitrekent en het resultaat op het scherm afdrukt.
d. Schrijf een C++-functie `void hoeveel (int n, int & aantal)` die het aantal factoren 2 in `n` bepaalt en in de parameter `aantal` stopt. Voor `n` gelijk aan 56 zou dit 3 zijn, want $56 = 2^3 \times 7$, voor `n` gelijk aan 7 zou dit 0 zijn (want 7 is niet deelbaar door 2) en voor 1024 zou dit 10 zijn, want $1024 = 2^{10}$.

2. We hebben een array `A` (`double A[n]`, met `const int n = 1234;`) met `n` *verschillende* getallen.
a. Geef een C++-functie `void wissel (double A[n], int i, int j)` die de waardes van de array-elementen `A[i]` en `A[j]` verwisselt (bij vaste `i` en `j` met $\mathtt{0} \leq \mathtt{i}, \mathtt{j} \leq \mathtt{n-1}$).
b. Geef een C++-functie `int linzoek(double A[n], double X)` die de array-index `i` met `X` gelijk aan `A[i]` teruggeeft. Gebruik *lineair zoeken*. Neem aan dat `X` in het array voorkomt.
c. Schrijf een C++-functie `void ruilen (double A[n], double X, double Y)` die de twee getallen `X` en `Y` in het array `A` van plaats verwisselt. Neem aan dat ze beide in `A` voorkomen. Gebruik de functies van **a** en **b**; het kan in één regel.
d. Leg (kort) in woorden uit hoe *binair zoeken* werkt, wanneer het toegepast kan worden, en waarom het in dat geval beter is dan lineair zoeken.

3. **a.** Bij een functie kun je te maken hebben met *call by value* en *call by reference*, en ook met *locale* en *globale* variabelen. Leg deze vier begrippen in woorden kort en duidelijk uit.
b. Een zeker C++-programma bevat de volgende programmaregels:

```
void draai (int & u, int & v) {
  u = u + v;  v = u - v;  u = u - v;
}  // draai
void jaar2002 (int & i, int grens) {
  int k; int j = 10;
  for ( k = 1; k <= grens; k++ ) {
    draai (i,j); i++;
    cout << i << " , " << j << endl;
  }  // for
  cout << i << " , " << j << " , " << k << " , " << grens << endl;
}  // jaar2002
```

Wat doet de functie `draai` met zijn parameters `u` en `v`? Hint: probeer eens `u = 5` en `v = 8`.
c. Wat levert op (met globale variabelen `a` en `grens` van type `int`):

```
a = 1998; grens = 5;
jaar2002 (a,grens);
cout << a << " , " << grens << endl;
```

Wat wordt er afgedrukt? Geef hierbij uiteraard uitleg.
d. Als **c**, maar nu zonder de drie `&`'s in de headings van `draai` en `jaar2002`.
e. Stel dat je in de functie `draai` de functie `jaar2002` zou willen aanroepen, bijvoorbeeld door de eerste regel te vervangen door `jaar2002 (u,v);`. Mag dat volgens de regels van C++? (Het gaat er niet om wat het programma dan eventueel zou doen.)

4. Deze som gaat over *magische vierkanten*. Deze zitten in een 2-dimensionaal array `int A[n][n];` met `n` rijen en `n` kolommen. Een voorbeeld met `n = 3`:

```
6  1  8
7  5  3
2  9  4
```

In een magisch vierkant staan de getallen $1, 2, \ldots, n^2$, en hebben alle rijen en alle kolommen dezelfde som (in het voorbeeld 15).

a. Schrijf een C++-functie `int rijsom (int A[n][n], int i)` die de som van de getallen uit de `i`-de rij van het array `A` uitrekent en teruggeeft.
b. Schrijf een C++-functie `void okeerijen (int A[n][n])`, die controleert of alle rijen van `A` dezelfde som hebben. Het resultaat wordt op het scherm afgedrukt: "Rijsommen gelijk" of "Rijsommen ongelijk". Gebruik de functie van **a**.
c. Schrijf een C++-functie `bool diag (int A[n][n])` die kijkt of de twee hoofddiagonalen (van links boven naar rechts onder, in het voorbeeld met `6`, `5` en `4`, en van rechts boven naar links onder, in het voorbeeld met `8`, `5` en `2`) dezelfde som hebben. In het voorbeeld is dat overigens zo, en zou de functie `true` op moeten leveren.
d. Schrijf een C++-functie `bool allegetallen (int A[n][n])` die controleert of alle getallen uit $1, 2, \ldots, n^2$ in het array voorkomen. Als dat zo is, moet de functie `true` teruggeven, en anders `false`. Er mag gratis (dat wil zeggen: zonder hem zelf te hoeven schrijven) gebruik gemaakt worden van een functie die checkt of een gegeven getal `X` in `A` voorkomt.

Uitwerkingen woensdag 3 april 2002

OPGAVE 1

```
a. int max2 (int x, int y) {
     if ( x > y )
       return x;
     else
       return y;
   }  // max2
b. int max4 (int x, int y, int u, int v) {
     return max2 (max2 (x,y), max2 (u,v));
   }  // max4
```

```
c. void som (int n) {
     int i; int res = 0;
     for ( i = 1; i < n; i++ )
       res = res + i * (i+1);
     cout << "Resultaat is: " << res << endl;
   }  // som
d. void hoeveel (int n, int & aantal) {
     aantal = 0;
     while ( n % 2 == 0 ) {
       n = n / 2;
       aantal++;
     }  // while
   }  // hoeveel
```

OPGAVE 2

```
a. void wissel (double A[n], int i, int j) {
     double temp = A[i];
     A[i] = A[j];
     A[j] = temp;
   }  // wissel
b. int linzoek (double A[n], double X) {
     int i = 0;
     while ( A[i] != X )
       i++;
     return i;
   }  // linzoek
c. void ruilen (double A[n], double X, double Y) {
     wissel (A,linzoek (A,X),linzoek (A,Y));
   }  // ruilen
```

d. Binair zoeken werkt alleen indien het array (oplopend) gesorteerd is. Het doet dan maximaal circa log n vergelijkingen in plaats van (maximaal) n voor lineair zoeken.
Je vergelijkt de gezochte X eerst met het (ongeveer) middelste array-element, stopt als dat X is, en als X kleiner is ga je in het "linker" array-gedeelte verder, anders in het "rechter" gedeelte; enz.

OPGAVE 3

a. Globale variabelen gelden in het gehele programma, en worden helemaal bovenin aangemaakt. Locale variabelen gelden (tijdelijk) alleen in de functie waarin ze aangemaakt zijn.
Variabelen kunnen call by value en call by reference worden meegegeven aan een functie. Bij call by value gaat alleen de waarde van de parameter naar de functie, alwaar een locale variabele deze waarde opvangt, en er met deze locale variabele wordt verder gerekend.
De oorspronkelijk variabele behoudt zijn waarde. Bij call by reference gaat als het ware de variabele zelf naar de functie, en kan dan ook blijvend veranderd worden. Eigenlijk wordt het adres (de reference)

doorgegeven.

b. De functie draai verwisselt (de inhouden van) u en v.

c. De for-loop uit jaar2002 wordt 5 maal doorlopen. Afgedrukt wordt:
11 , 1998 1999 , 11 12 , 1999 2000 , 12 13 , 2000
13 , 2000 , 6 , 5 13 , 5

d. Nu doet draai niets meer met u en v! Afgedrukt wordt:
1999 , 10 2000 , 10 2001 , 10 2002 , 10 2003 , 10
2003 , 10 , 6 , 5 1998 , 5

e. Dat mag niet, want dan zou jaar2002 boven draai moeten staan. Als je het echt wilt moet je een prototype van jaar2002 (alleen de kopregel met een ; er achter) boven draai zetten. Je krijgt dan wel ingewikkelde recursie overigens!

OPGAVE 4

a.
```
int rijsom (int A[n][n], int i) {
  int som = 0; int j;
  for ( j = 0; j < n; j++ )
    som = som + A[i][j];
  return som;
} // rijsom
```

b.
```
void okeerijen (int A[n][n]) {
  int som = rijsom (A,0);
  bool okee = true; int i;
  for ( i = 1; i < n; i++ )
    if ( som != rijsom (A,i) )
      okee = false;
  if ( okee )
    cout << "Rijsommen gelijk" << endl;
  else
    cout << "Rijsommen ongelijk" << endl;
} // okeerijen
```

c.
```
bool diag ( int A[n][n]) {
  int som1 = 0, som2 = 0; int i;
  for ( i = 0; i < n; i++ ) {
    som1 = som1 + A[i][i];
    som2 = som2 + A[i][n-1-i];
  } // for
  return ( som1 == som2 );
} // diag
```

d.
```
bool allegetallen (int A[n][n]) {
  int X;
  for ( X = 1; X <= n*n; X++ )
    if ( ! komtvoor (A,X) )
      return false;
  return true;
} // allegetallen
```

5.2 Tentamen vrijdag 28 juni 2002

De opgaven tellen alle vier even zwaar mee. Veel succes!

1. a. Schrijf een C++-functie `double gem2 (double x, double y)` die het gemiddelde van de twee getallen `x` en `y` teruggeeft (met behulp van een `return`-statement).
b. Schrijf een C++-functie `double gem4 (double x, double y, double u, double v)` die het gemiddelde van de vier getallen `x`, `y`, `u` en `v` teruggeeft. De functie moet gebruik maken van de functie `gem2` van **a**; het kan in één regel.
c. Schrijf een C++-functie `int wortel (double x)` die de wortel uit het positieve getal `x` als volgt benadert. Het grootste gehele getal waarvan het kwadraat nog kleiner dan of gelijk aan `x` is moet worden geretourneerd. Gebruik een `while`-loop.
d. Stel we hebben een array `woordje` met `5` letters (`char woordje[5]`) en een array `verhaal` met `100` letters (`char verhaal[100]`). We willen weten of het `woordje` in het `verhaal` voorkomt: op `5` direct opeenvolgende posities moeten de twee arrays precies overeenstemmen. Schrijf een boolese C++-functie die dit controleert.

2. We hebben een array `A` (`double A[n]`, met `const int n = 1234;`) met `n` *verschillende* getallen.
a. Geef een C++-functie `void wissel (double A[n], int i, int j)` die de waardes van de array-elementen `A[i]` en `A[j]` verwisselt (bij vaste `i` en `j` met $0 \leq$ `i`, `j` $\leq$ `n-1`).
b. Geef een C++-functie `int grootste (double A[n], int i, int j)` die het grootste getal uit `A[i]`, `A[i+1]`, ..., `A[j]` opzoekt, en de desbetreffende array-index teruggeeft.
c. Schrijf een C++-functie `void sorteer (double A[n])` die het array `A` oplopend sorteert door herhaald de functies van **a** en **b** aan te roepen.
d. Leg kort in woorden uit hoe *bubblesort* werkt.

3. a. Bij een functie kun je te maken hebben met *call by value* en *call by reference*, en ook met *locale* en *globale* variabelen. Leg deze vier begrippen in woorden kort en duidelijk uit.
b. Een zeker C++-programma bevat de volgende programmaregels:

```
int brom (int & x) {
  x = x + 5; return x;
} // brom
void hmm (int a, int b, int & c ) {
  int z = 6;
  c = a + b + brom (z);
  b = a + b; a = b - a; b = b - a;
  z++;
  cout << a << b << c << z << endl;
}  // hmm
```

Wat levert op (met `x`, `y` en `z` van type `int`;):

```
x = 1; y = 3; z = 5; hmm (x+y,y-x,z); cout << x << y << z << endl;
```

Wat wordt er afgedrukt? Geef hierbij uiteraard uitleg.
c. Idem, maar nu zonder de `&` in de heading van `brom` en zonder die in de heading van `hmm` (beide dus weggelaten).
d. Wat levert op (maar nu met `&`-s voor *alle* parameters in `brom` en `hmm` (4 stuks)):

```
x = 1; y = 3; z = 5; hmm (x,x,x); cout << x << y << z << endl;
```

e. Mag ergens in het programma de aanroep `brom (brom (y))` voorkomen? Leg uit.

4. Deze som gaat over een *afstandentabel*. Deze zit opgeslagen in een 2-dimensionaal array `int A[n][n];` met `n` rijen en `n` kolommen. Een voorbeeld met `n = 3`:

```
 0  3 18
 3  0  7
18  7  0
```

Het getal in de `i`-de rij, `j`-de kolom geeft de afstand aan tussen de plaatsen `i` en `j`: de afstand tussen `0` en `2` is `18`.

a. Schrijf een C++-functie `int verste (int A[n][n], int& i, int& j)` die de twee het verst uit elkaar gelegen plaatsen opzoekt in het array `A` en deze in de call-by-reference parameters `i` en `j` stopt, met in `i` het kleinste rangnummer, en hun afstand retourneert. In het voorbeeld zijn het `0` en `2` (afstand `18`).

b. Normaal is het zo dat als je rechtstreeks van `i` naar `j` reist, dit korter is dan als je via `k` gaat. Schrijf een C++-functie `void driehoek (int A[n][n])`, die controleert of dit voor alle drietallen plaatsen `i`, `j` en `k` in `A` klopt. Het resultaat wordt op het scherm afgedrukt: "klopt" of "klopt niet". In het voorbeeld klopt het niet: van `0` naar `2` via `1` is samen `3` plus `7`, oftewel `10`, en rechtstreeks is maar liefst `18`.

c. Stel je doet het volgende. Je begint in plaats `i`. Ga nu naar de plaats `i+1` (naar `0` als je in plaats `n-1` zat), enzovoorts. Dit doe je alleen als de lengte van deze stap groter is dan de vorige stap (voor de eerste stap is dit automatisch goed). Schrijf een C++-functie `int ga (int A[n][n], int i)` die uitrekent hoever je dan in totaal bent gekomen. Gebruik een `while`-loop. In het voorbeeld, met `i` gelijk aan `1`: vanuit `1` doe je een stap van `7` naar `2`, vanuit `2` een stap van `18` naar `0`, en daar blijf je: de stap van `0` naar `1` ter grootte `3` is kleiner dan `18`; totaal gelopen: `25`.

Uitwerkingen vrijdag 28 juni 2002

OPGAVE 1

```
a. double gem2 (double x, double y) {
     returm ( x + y ) /2;
   }  // gem2
b. double gem4 (double x, double y, double u, double v) {
     return gem2 (gem2 (x,y), gem2 (u,v));
   }  // gem4
c. int wortel (double x) {
     int root = 0;
     while ( root * root <= x )
       root++;
     return (root - 1);
   }  // wortel
d. bool controle (char woordje[5], char verhaal[100]) {
     int i, j;
     bool okee;
     for ( i = 0; i < 100-5+1; i++ ) {
       okee = true;
```

```
      for ( j = 0; j < 5; j++ )
        if ( woordje[j] != verhaal[i+j] )
          okee = false;
      if ( okee )
        return true;
    }  // for
    return false;
  }  // controle
```

OPGAVE 2

```
a. void wissel (double A[n], int i, int j) {
    double temp = A[i];
    A[i] = A[j];
    A[j] = temp;
  }  // wissel
b. int grootste (double A[n], int i, int j) {
    int k = i;
    for (l = i+1; l <= j; l++ )
      if ( A[l] > A[k] )
        k = l;
    return k;
  }  // grootste
c. void sorteer (double A[n]) {
    int k, m;
    for ( m = n-1; m > 0; m-- ) {
      k = grootste (A,0,m);
      wissel (A,k,m);
    }  // for
  }  // sorteer
```

d. Loop herhaald door de rij heen. Bij het lopen vergelijk je elk buurpaar en wisselt de twee getallen om als ze verkeerd om staan. (Tijdens iedere ronde borrelt het grootste getal naar achteren; je kunt dus iedere ronde steeds een paar eerder stoppen.)

OPGAVE 3

a. Globale variabelen gelden in het gehele programma, en worden helemaal bovenin aangemaakt. Locale variabelen gelden (tijdelijk) alleen in de functie waarin ze aangemaakt zijn.
Variabelen kunnen call by value en call by reference worden meegegeven aan een functie. Bij call by value gaat alleen de waarde van de parameter naar de functie, alwaar een locale variabele deze waarde opvangt, en er met deze locale variabele wordt verder gerekend.
De oorspronkelijk variabele behoudt zijn waarde. Bij call by reference gaat als het ware de variabele zelf naar de functie, en kan dan ook blijvend veranderd worden. Eigenlijk wordt het adres (de reference) doorgegeven.

b. 2 4 17 12 1 3 17

```
c. 2 4 17 7    1 3 5
d. 0 0 0 12    0 3 5
e. Dat mag alleen als er GEEN & voor de parameter
   x van de functie brom staat.

OPGAVE 4
a. int verste (int A[n][n], int& i, int& j) {
     int gr = 0, r, s;
     for ( r = 0; r < n; r++ )
       for ( s = r+1; s < n; s++ )
         if ( A[r][s] > gr ) {
           gr = A[r][s];
           i = r;
           j = s;
         }  // if
     return gr;
   }  // verste
b. void driehoek (int A[n][n]) {
     int i, j, k;
     bool okee = true;
     for ( i = 0; i < n; i++ )
       for ( j = 0; j < n; j++ )
         for ( k = 0; k < n; k++ )
           if ( A[i][j] > A[i][k] + A[k][j] )
             okee = false;
     if ( okee )
       cout << "klopt" << endl;
     else
       cout << "klopt niet" << endl;
   }  // driehoek
c. int ga (int A[n][n], int i) {
     int tel = 0, vorige = 0;
     while ( A[i][(i+1)%n] > vorige ){
       tel = tel + A[i][(i+1)%n];
       vorige = A[i][(i+1)%n];
       i = (i+1)%n;
     }  // while
     return tel;
   }  // ga
```

5.3 Tentamen maandag 26 augustus 2002

De opgaven tellen alle vier even zwaar mee. Veel succes!

1. **a.** Schrijf een C++-functie `double middel (double x, double y, double z)` die het *middelste in grootte* van de drie verschillende getallen `x`, `y` en `z` teruggeeft (met behulp van een `return`-statement). Voorbeeld: bij 3,7,2 is 3 het resultaat.
b. Schrijf een C++-functie `int fac (int n)` die `n!` $= n \times (n-1) \times \cdots \times 3 \times 2 \times 1$ berekent.

c. Schrijf een C++-functie `int binom (int n, int k)` die de binomiaalcoefficient

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

uitrekent. Gebruik **b**.
d. Stel we hebben een array `woordje` met `6` letters (`char woordje[6]`) en een array `verhaal` met `100` letters (`char verhaal[100]`). We willen weten of het `woordje` ergens in het `verhaal` voorkomt: op `6` direct opeenvolgende posities moeten de twee arrays precies overeenstemmen. Schrijf een boolese C++-functie die dit controleert.

2. We hebben een array `A` (`double A[n]`, met `const int n = 1234;`) met `n` getallen.
a. Geef een C++-functie `void wissel (double A[n], int i, int j)` die de waardes van de array-elementen `A[i]` en `A[j]` verwisselt (bij vaste `i` en `j` met `0` $\leq$ `i, j` $\leq$ `n-1`).
b. Geef een C++-functie `void sorteer (double A[n])` die het array `A` oplopend sorteert met *bubblesort*. Gebruik de functie van **a**.
c. Hoeveel vergelijkingen tussen array-elementen doet het algoritme van **b**?
d. Stel dat het array `A` oplopend gesorteerd is. We maken één willekeurig array-element `3.14`, en verstoren daarmee waarschijnlijk de sortering. We willen `A` opnieuw, en met weinig inspanning, sorteren. Geef kort *in woorden* een methode aan die dit doet.

3. **a.** Bij een functie kun je te maken hebben met *call by value* en *call by reference*, en ook met *locale* en *globale* variabelen. Leg deze vier begrippen in woorden kort en duidelijk uit.
b. Een zeker C++-programma bevat de volgende programmaregels:

```
void abcde (int & x, int & y, int & z) {
  int temp = x; x = y; y = z; z = temp; x--; y--; z--;
  cout << x << y << z << endl;
}  // abcde
int fghij (int & x, int & y, int & z) {
  if ( z > x ) abcde (y,z,x); else abcde (x,y,z);
  cout << z << x << y << endl; return x + z;
}  // fghij
```

Wat levert op (met globale variabelen `x`, `y` en `z` van type `int`):

```
x = 4; y = 17; z = 25;
abcde (x,y,z); cout << fghij (x,y,z) << endl;
cout << x << y << z << endl;
```

Wat wordt er afgedrukt? Geef hierbij uiteraard uitleg.
c. Als **b**, maar met weglating van de zes `&`'s.
d. Als **b.** (dus met alle zes `&`'s er bij!), voor

```
x = 10; y = 12; z = 15;
abcde (x,x,x); cout << fghij (x,x,x) << endl;
cout << x << y << z << endl;
```

e. Mag ergens in het programma de aanroep `abcde (fghij (x),x,x)` voorkomen? Leg uit.

4. Deze som gaat over een 2-dimensionaal array `int A[n][n];` met `n` rijen en `n` kolommen. Twee voorbeelden met `n = 3`:

```
 1  4  0      2  4  1
16  6  7     16  7  7
 2 24 11      2 24 11
```

a. Schrijf een C++-functie `int aantal (int A[n][n], int X)` die bepaalt hoevaak het getal `X` in het array `A` voorkomt. In het rechter array komt `7` twee keer voor.
b. Schrijf een C++-functie `bool uniek (int A[n][n])` die bepaalt alle getallen uit `A` precies één keer voorkomen; zo ja, dan moet de functie `true` retourneren, en anders `false` Voor de linker matrix uit het voorbeeld moet het `true` zijn, voor de rechter `false`. Gebruik **a**.
c. Schrijf een C++-functie `void doe (int A[n][n], int i, int j)` die de onmiddellijk aangrenzende horizontale en verticale buren van het getal uit de `i`-de rij, `j`-de kolom, met `1` ophoogt. In het voorbeeld ontstaat de rechter uit de linker matrix door de aanroep `doe (A,0,1)`: de drie buren van `4` worden opgehoogd.
d. Schrijf een C++-functie `void verander (int A[n][n])` die één voor één de elementen op de hoofddiagonaal van `A` afloopt, te beginnen links boven (in de linker matrix van het voorbeeld bestaat de hoofddiagonaal uit `1`, `6` en `11`), en telkens de functie `doe` van **c** aanroept. De functie moet stoppen als niet alle elementen van `A` uniek meer zijn (gebruik **b**) of als de hele hoofddiagonaal is behandeld.

Uitwerkingen Tentamen maandag 26 augustus 2002

OPGAVE 1

```
a. double middel (double x, double y, double z) {
     if ( ( x < y && y < z ) || ( z < y && y < x ) ) return y;
     else if ( ( y < x && x < z ) || ( z < x && x < y ) ) return x;
     else return z;
   }  // middel
b. int fac (int n) {
     int i, res = 1;
     for ( i = 1; i <= n; i++ )
       res = res * i;
     return res;
   }  // fac
c. int binom (int n, int k) {
     return fac (n) / ( fac (k) * fac (n-k) );
   }  // binom
d. bool controle (char woordje[6], char verhaal[100]) {
     int i, j;
     bool okee;
     for ( i = 0; i < 100-6+1; i++ ) {
       okee = true;
       for ( j = 0; j < 6; j++ )
```

```
        if ( woordje[j] != verhaal[i+j] )
          okee = false;
      if ( okee )
        return true;
    }  // for
    return false;
  }  // controle
```

OPGAVE 2

a.
```
void wissel (double A[n], int i, int j) {
    double temp = A[i];
    A[i] = A[j];
    A[j] = temp;
  }  // wissel
```
b.
```
void sorteer (double A[n]) {
    int i, ronde;
    for ( ronde = 1; ronde < n; ronde++ )
      for ( i = 0; i < n-ronde; i++ )
        if ( A[i] > A[i+1] )
          wissel (A,i,i+1);
  }  // sorteer
```
c. (n-1) + (n-2) + ... + 3 + 2 + 1 = n (n-1) / 2

d. Doe EEN bubblesort-ronde naar rechts, en dan EEN naar links.

OPGAVE 3

a. Globale variabelen gelden in het gehele programma, en worden helemaal bovenin aangemaakt. Locale variabelen gelden (tijdelijk) alleen in de functie waarin ze aangemaakt zijn.
Variabelen kunnen call by value en call by reference worden meegegeven aan een functie. Bij call by value gaat alleen de waarde van de parameter naar de functie, alwaar een locale variabele deze waarde opvangt, en er met deze locale variabele wordt verder gerekend.
De oorspronkelijk variabele behoudt zijn waarde. Bij call by reference gaat als het ware de variabele zelf naar de functie, en kan dan ook blijvend veranderd worden. Eigenlijk wordt het adres (de reference) doorgegeven.

b. 16 24 3 23 2 15 15 23 2 38 23 2 15

c. 16 24 3 24 3 16 25 4 17 29 4 17 25

d. 7 7 7 4 4 4 4 4 4 8 4 12 15

e. Ja, mits de x-parameter (de eerste) van fghij call by value is.

OPGAVE 4

a.
```
int aantal (int A[n][n], int X) {
    int tel = 0, i, j;
    for ( i = 0; i < n; i++ )
      for ( j = 0; j < n; j++ )
        if ( A[i][j] == X )
```

```
          tel++;
    return tel;
  }  // aantal
b. bool uniek (int A[n][n]) {
    int i, j;
    for ( i = 0; i < n; i++ )
      for ( j = 0; j < n; j++ )
        if ( tel (A,A[i][j]) != 1 )
          return false;
    return true;
  }  // uniek
c. void doe ( int A[n][n], int i, int j) {
    if ( i-1 >= 0 ) A[i-1][j]++;
    if ( i+1 < n ) A[i+1][j]++;
    if ( j-1 >= 0 ) A[i][j-1]++;
    if ( j+1 < n ) A[i][j+1]++;
  }  // doe
d. void verander (int A[n][n]) {
    int i = 0;
    while ( i < n && uniek (A) ) {
      doe (A,i,i);
      i++;
    }  // while
  }  // verander
```