OPDRACHTEN

PROGRAMMEREN

(5e en 6e kwartaal natuur- en sterrenkunde)

July 26, 2012

Faculteit der Natuurwetenschappen, Wiskunde en Informatica
P.F. Klok
J.M. Thijssen
W.L. Meerts

Radboud Universiteit Nijmegen

1 Inleiding

Het doel van Programmeren is om je bekend te maken met de computer en om je, aan de hand van vooral fysische voorbeelden, te leren op een algoritmische manier problemen op te lossen. In deze opdrachten gaat het erom zulke algoritmen in de programmeertaal C te verwezenlijken.
Naast fysische problemen komen ook algemene zaken aan de orde die belangrijk zijn in de informatica, zoals het werken met teksten en sorteren van gegevens.

In de moderne fysica speelt de computer een steeds grotere rol: in vrijwel alle gebieden van de fysica wordt de computer toegepast en dit gebeurt op een zeer veelzijdige manier.
Men kan de toepassingen als volgt indelen:

- **Numerieke analyse:** het getalmatig oplossen van vergelijkingen waar analytische methoden tekort schieten of onwerkbaar ingewikkeld zijn.
- **Besturing en gegevensverwerking** bij experimentele opstellingen.
- **Simulaties** van modellen om experimenten te verklaren of om de modellen zelf beter te begrijpen.
- **Grafische toepassingen:** visualiseren van processen of van numeriek bepaalde oplossingen van vergelijkingen.
- **Algebraïsche manipulatie:** *symbolisch,* dus niet getalmatig, verwerken van vergelijkingen.

In de opdrachten zullen, naast niet specifiek fysische problemen, de eerste vier toepassingsgebieden aan de orde komen.

Niet alle opdrachten dienen ingeleverd te worden. Op het college wordt een lijst verstrekt met in te leveren opdrachten. Alleen de in te leveren opdrachten tellen mee in de eindbeoordeling.

2 Opdrachten

1. **Verwisselen**
   Schrijf een programma dat twee **float** variabelen, **a** en **b**, vult met twee vanaf het toetsenbord in te lezen getallen en dat vervolgens de inhoud van **a** en **b** verwisselt en op het scherm afdrukt:

   (a) m.b.v. invoering van een hulpvariabele **c**,

   (b) zonder invoering van een hulpvariabele.

2. **Meetkunde**
   Schrijf een programma dat vraagt om een hoek (in *graden*) en de lengte van de twee aanliggende zijden van een driehoek en dat vervolgens de lengte geeft van de overstaande zijde en de cosinus van de twee overige hoeken.
   Gebruik hierbij o.a. de standaardfunctie **cos**:

   cos (x)

   die de cosinus van **x** levert, waarbij **x** in *radialen* gegeven is.

3. **Vierkantsvergelijking**
   Schrijf een programma dat aan de gebruiker vraagt om de coëfficienten **a**, **b** en **c** op te geven van de kwadratische vergelijking voor de variabele **x**:

   $$ax^2 + bx + c = 0$$

   en vervolgens de oplossingen van deze vergelijking op het scherm afdrukt.
   Houd rekening met complexe oplossingen en met het nul zijn van de parameters **a**, **b** en/of **c**!
   Gebruik de standaard functie **sqrt**:

   ```
   sqrt (x)
   ```

   die $\sqrt{x}$ levert als $x \geq 0$.

4. **Priemgetallen**
   Schrijf een programma dat alle priemgetallen vindt tot een gegeven maximale waarde met behulp van de Zeef van Eratosthenes.

   Zeef van Eratosthenes (bibliothecaris van Alexandrië vanaf ca. 240 v.Chr.) is een al zeer lang bekend algoritme om priemgetallen te vinden. Deze elegante methode is vooral efficiënt wanneer hij wordt gebruikt voor de kleinere priemgetallen. De methode vergt echter het bijhouden van alle getallen kleiner dan de gebruikte bovengrens, wat naarmate de te bepalen priemgetallen groter worden een steeds groter nadeel wordt.

   (1) Maak een gesorteerde lijst van alle getallen van 2 tot een zelf te kiezen maximum.

   (2) Kies het kleinste getal uit de lijst.

   (3) Streep alle veelvouden van het gekozen getal door (maar niet het getal zelf).

   (4) Kies het volgende getal uit de lijst en ga verder met stap 3.

   De getallen die op deze manier overblijven zijn alle priemgetallen tot het maximum.

   De procedure kan op enkele manieren versneld worden.

   - Het heeft geen zin in stap 4 een getal te kiezen dat al doorgestreept is, want alle veelvouden daarvan zijn al doorgestreept.
   - Men kan met doorstrepen beginnen met het kwadraat van het gekozen getal. Alle kleinere veelvouden zijn al doorgestreept.
   - Is het gekozen getal groter dan de wortel uit het maximum, dan is de procedure voltooid.

   Hint: Definieer een array prime[] met een lengte van maximum priemgetal+1 (maxpriem). Zet alle begin waardes op 1 en voer bovenstaande zeef uit (i = 1, maxpriem). Als het i geen priemgetal is zet prime[i]=0.

5. **Kalender**
Schrijf een programma dat aan de gebruiker vraagt om een datum (dag, maand en jaar) op te geven en dat vervolgens op het scherm schrijft welke dag van de week (maandag, ... zondag) bij die datum hoort. Zorg dat het programma in ieder geval voor de 20ste en de 21ste eeuw werkt.
NB 1: **Schrikkeljaren** zijn de jaren die deelbaar zijn door 4, **uitgezonderd** de eeuwjaren die niet deelbaar zijn door 400.
NB 2: Het is niet de bedoeling om een analytische formule die je bijvoorbeeld via Google kunt vinden te gebruiken. Bepalen van de dag van de week via *if, switch* .. ed.

6. **Sorteren**
Sorteren van getallen.

   (a) Maak een programma dat eerst vraagt hoeveel getallen je wilt sorteren en vervolgens dit opgegeven aantal inleest vanaf het toetsenbord. Sorteer de getallen in aflopende volgorde en print het resultaat naar het scherm. Maak gebruik van dynamisch alloceren (*malloc*) om een array voor de in te lezen waardes te alloceren.

   (b) Maak nu een programma dat een aantal getallen direct inleest zonder van te voren te weten hoeveel getallen er komen en deze in een array plaatst. Maak gebruik van *malloc* en *realloc*. Bedenk een criterium om het einde van de invoer aan te geven. Sorteer vervolgens de ingelezen getallen in oplopende volgorde en print deze weer uit.
   NB: Je kunt natuurlijk bij elke nieuw ingevoerde waarde *realloc* gebruiken om het reeds gebruikte geheugen allocatie met één te verhogen. Dit is echter zeer inefficient. Beter is om het reeds gedeclareerde geheugen uit te breiden met een flink aantal plaatsen (blok van 1000 of zoiets) of het te verdubbelen. Je moet dan nog wel een extra teller bijhouden!

7. **String manipulatie**

Schrijf een programma dat het aantal woorden in een ingegeven string telt. Woorden worden van elkaar gescheiden door een of meerder spaties, leestekens of combinatie van leestekens en spatie(s). Het programma moet daar rekening mee houden.

Maak gebruikt van de C-string functies (Sectie B3 van Brian W. Kernighan, Dennis M. Ritchie, *The C Programming Language, second edition*)

8. **Interpolatie**
We willen het nulpunt x_0 van een functie f gaan bepalen. Het nulpunt wordt gevonden met een bepaalde nauwkeurigheid: dit kan de nauwkeurigheid zijn waarbinnen f van nul mag afwijken, òf waarmee de gevonden waarde van x van x_0 mag afwijken. De onderdelen (a) en (b) dienen als voorbereidings stappen voor de uiteindelijk opgave in (c). **Alleen onderdeel (c) moet ingeleverd worden.**

   (a) Beschouw eerst (strikt) stijgende functies op het interval $[0, 1]$.
   Bedenk een methode waarmee het mogelijk is om met n stappen x_0 met een nauwkeurigheid van 2^{-n} te bepalen. Schrijf een procedure die dit algoritme uitvoert.
   Test deze procedure voor $f(x) = x^2 + 3x - 1$.

   Hint: Deel het interval door midden en kijk of de nuldoorgang links of rechts hiervan ligt. Herhaal dit n-maal op het interval waar de nul doorgang gevonden is.

   (b) Idem, voor functies die òf strikt stijgend òf strikt dalend zijn op $[0, 1]$.
   Test deze procedure voor $f(x) = -x^2 - 3x + 1$.

   (c) Idem, voor functies die meerdere nulpunten hebben op $[0, 1]$, die onderlinge afstand $> \Delta$ hebben (neem aan dat er geen dubbele nulpunten zijn).
   Test deze procedure voor $f(x) = (x - \frac{1}{4})(x - \frac{1}{2})(x - \frac{3}{4})$. Bij deze procedure moet dus een schatting van Δ gemaakt worden door de gebruiker, aangezien de procedure zelf natuurlijk niet kan weten wat de minimale afstand tussen de nulpunten is.

9. **Lineaire fit**

De kleinste-kwadratenmethode wordt gebruikt om een goed passende rechte lijn door een aantal meetpunten te tekenen.
Beschouw een rij van N meetpunten (X_i, Y_i) met $X_i < X_j$ voor $i < j$.
Voor de beste fit $y = mx + b$ door de meetpunten geldt dan dat:

$$m = \frac{N\sum_i X_i Y_i - \sum_i X_i \sum_j Y_j}{N\sum_i X_i^2 - (\sum_i X_i)^2}$$

$$b = \frac{(\sum_i {X_i}^2)\sum_j Y_j - \sum_i X_i \sum_j X_j Y_j}{N\sum_i X_i^2 - (\sum_i X_i)^2}$$

Schrijf nu een programma dat een rij waarden (X_i, Y_i) fit aan bovenstaande formules en deze in een grafiek tekent. De verzameling van de punten (X_i, Y_i) mag je in een statische array in het programma zetten.
Maak bij deze opgave gebruik van GNUplot. Het C-programma dient het data file waar de te plotten punten in staan te genereren en tevens het script file voor GNUPlot. Teken in deze grafiek ook de rechte lijn die berekend is volgens bovenstaande formules.
NB. Je hebt nu de keuze om een los extern script file voor GNUPlot te maken daar resultaten van de fit in te substitueren **òf** het scriptfile direct vanuit je C-programma te creëren. Dit laatste geniet de voorkeur.

10. **Differentiaalvergelijkingen.**

Zeer veel fysische problemen worden geregeerd door een differentiaalvergelijking. In deze opdracht is het de bedoeling om zulke vergelijkingen op verschillende manieren numeriek op te lossen. Het resultaat dient op het scherm te worden getekend met behulp van GNUplot.

(a) We beginnen met de meest directe methode die we zullen toepassen op een eerste orde differentiaalvergelijking:

$$\begin{aligned} \tfrac{dx(t)}{dt} &= f(x) \\ x(t_0) &= x_0 \end{aligned}$$

Gevraagd wordt nu de waarde van $x(t)$ voor $t_0 \le t \le t_1$, voor zekere t_1.
De eenvoudigste schatting voor $x(t_0 + \Delta)$ krijgen we door aan te nemen dat $f(x)$ tussen t_0 en $t_0 + \Delta$ niet noemenswaardig verandert; dit geeft:

$$x(t_0 + \Delta) = x(t_0) + f(x_0) \cdot \Delta$$

Het zal duidelijk zijn dat deze benadering beter werkt naarmate Δ kleiner wordt.
Met de gevonden waarde voor $x(t_0+\Delta)$ bepalen we $x(t_0+2\Delta)$ enz., totdat we bij $x(t_1)$ zijn aangeland.
Schrijf een procedure die de differentiaalvergelijking op deze wijze oplost. Definieer f als een functie:

```
float f (float x)
```

De procedure heeft als parameters de linker- en rechtergrens van het interval waarover geïntegreerd wordt, de beginvoorwaarde x_0 en de stapgrootte Δ.
Gebruik deze procedure om de oplossing van de volgende beginwaardeproblemen op het scherm te tekenen:

$$\begin{aligned} \tfrac{dx(t)}{dt} &= -x(t), \qquad x(0) = 1 \quad t \in [0, 1] \\ \tfrac{dx(t)}{dt} &= \sqrt{(1 - x^2(t))}, \quad x(0) = 0 \quad t \in [0, \pi/2]. \end{aligned}$$

Controleer het gevonden resultaat.

(b) Met behulp van een soortgelijke methode als in het vorige onderdeel kunnen we tweede orde differentiaalvergelijkingen aan.
Schrijf een procedure die het volgende beginwaardenprobleem oplost:

$$\begin{aligned} &\tfrac{d^2x(t)}{dt^2} = f(x) \\ &x(t_0) = 0; \qquad \tfrac{dx(t_0)}{dt} = 1; \qquad t_0 = 0. \end{aligned}$$

Test deze procedure door bijvoorbeeld $f(x) = -ax$ met $a > 0$ te nemen. Dit is de harmonische oscillator. Plot het gevonden numerieke resultaat tegelijk met de analytische oplossing en bekijk het effect van de keuze van de stap grootte.

Hint: Los de tweede orde differentiaal vergelijking op door $v(t) = \frac{dx(t)}{dt}$ te definieren. Je moet dan de de volgende twee gekoppelde vergelijkingen oplossen.

$$\frac{dv(t)}{dt} = f(x)$$
$$\frac{dx(t)}{dt} = v(t)$$

Let op de volgorde bij het oplossen.

11. Random Walk

Maak een random walk door het twee dimensionale xy-vlak. Doe dit door telkens een random stap in de x en y richting te genereren. Maak hiervan een scatter plot m.b.v. GNUplot.

Hint: Gebruik bijvoorbeeld de bibliotheek functie rand() om een random getal tussen -1 en +1 te laten genereren en gebruik dit als stap grootte.

12. Matrices

Schrijf een programma dat twee matrices inleest vanuit twee aparte bestanden, de matrices met elkaar vermenigvuldigt als dit mogelijk is (rij maal kolom) en het resultaat afdrukt op het scherm èn wegschrijft in een bestand. De extra **eis** waaraan het programma moet voldoen is dat het van de commando regel (in een cmd-box, onder Windows) op de volgende manier aangeroepen moet worden:

multiply-matrices inputfile1 inputfile2 outputfile

Je hebt nu twee opties voor het inlezen van de matrices vanuit de bestanden:

(I) Dit is een minimale, snelle en meest simpele oplossing. Hierbij ga je ervan uit dat op de eerste regel van het input bestand het aantal rijen (N_{rij}) en kolommen (N_{kolom}) staat in een vast formaat. En vervolgens dat er N_{rij} regels volgen met op elke regel de N_{kolom} kolom waardes van de gegeven rij in een vast formaat, bijvoorbeeld de elementen gescheiden met **één** spatie.

(II) Meer uitdagend is de volgende optie. In dit geval is bij het openen van het bestand niet bekend hoeveel rijen en kolommen de matrix heeft. We weten alleen dat dat op elke regel een rij staat. De getallen kunnen gescheiden zijn door spatie(s) (meerdere achter elkaar mogelijk), een "," of een TAB; en wel door elkaar heen gebruikt! Begin en eind van de regel mag ook een of meerdere spaties bevatten. Wel moeten er op iedere regel natuurlijk evenveel kolom-waardes staan (check dit!). Deze optie geeft de gebruiker optimale vrijheid in het maken van zijn input bestand, maar eist uiteraard meer van de programeur.

Hints en eisen opdracht 12

Opdracht 12 kun je gemakkelijk een zooitje laten worden in je code. Daarom enkele aanwijzingen waarmee de opdracht hopelijk begrijpelijker en makkelijker uitvoerbaar wordt. Vanzelfsprekend hoef je niet alles – behalve de eisen – tot op de letter te volgen; we proberen slechts aan te geven hoe je het jezelf wat eenvoudiger kunt maken.

Eisen

Zoals al in de opdracht staat moet je programma de volgende argumenten aannemen:

```
program <input1> <input2> <output>
```

Ter herinnering, in het college is al uitgelegd dat je daarvoor de argumenten van `main()` dient te gebruiken:

```
int main(int argc, char **argv) {
}
```

Concreet betekent dat, dat wij ons programma dus als volgt willen kunnen aanroepen (dit voorbeeld is op een Linux-machine, de gekozen bestandsnamen zijn uiteraard arbitrair):

```
./multiply-matrices inputfile1 inputfile2 outputfile
```

Opslag

Om je matrices op te slaan, is het handig gebruik te maken van een `struct`, bijvoorbeeld een van de volgende vorm:

```
typedef struct {
    float** data;
    int     nRows;
    int     nCols;
} Matrix;
```

Hiermee is je `struct` direct gedefinieerd (via de `typedef`) als een nieuw type: `Matrix`.

Deel I

Deel I is eigenlijk een speciaal geval van deel II. Alle functies – en het type `Matrix` – zijn dus ook nuttig voor dit onderdeel! Het enige dat je *niet* nodig hebt is het bepalen van de grootte van de ingelezen matrices; in dit onderdeel mag je er van uit gaan dat de matrices er netjes uit zien.

Samengevat: schrijf hier je code dus op een dusdanige manier, dat je bij deel II alleen maar de grootte van de matrix hoeft te bepalen en voor de rest vrijwel alles kunt hergebruiken!

Gebruik de functie *fscanf* voor het geformateerd inlezen van de eerste regel. Dit geeft de grootte van de matrix. Lees de volgende regels in met *fgets*.

Deel II

Matrixgrootte bepalen

Je moet een matrix lezen en alloceren van onbekende grootte. De opdracht geeft al aan dat het onvermijdelijk is de datafile tweemaal te lezen: eerst om te bepalen wat voor matrix je hebt, en vervolgens om hem daadwerkelijk uit te lezen. *Gebruik hiervoor functies!*

Om de grootte van een matrix te bepalen (en erna het geheugen te reserveren) kun je een functie als volgt definiëren:

```
Matrix buildMatrix(const char*);
```

Om de grootte te bepalen zijn de volgende zaken nuttig om je te realiseren:

(1) je kunt je data regel voor regel uitlezen;

(2) een matrix die er normaal uitziet heeft op elke *niet-lege* regel evenveel getallen;

(3) eventuele spaties, tabs, komma's, newlines (`\n`) en carriage returns (`\r`) doen er niet toe;

(4) een getal is dus een blokje niet-whitespace.

Om te bepalen hoeveel getallen je hebt kun je dat per regel doen. (Hint: functie!) Een goede mogelijkheid hiervoor is je regel te *tokenizen* (i.e. ophakken in blokjes gescheiden door een aantal delimiters). In C is de functie daarvoor `strtok()`. De delimiters zijn in ons geval spatie(s), ",", TAB (\t), LineFeed (\n) (Linux) en mogelijkerwijs **ook** een CarriageReturn (\r) (DOS).

Matrix inlezen

De matrix inlezen is nu triviaal: `fscanf()` slaat alle whitespace over tot het volgende getal dat geconverteerd kan worden naar `float`. Je hebt dus voldoende aan een functie als:

```
void readMatrix(const char*, Matrix*);
```

Matrices vermenigvuldigen

Gebruik ook hier weer een functie; definieer er bijvoorbeeld een als

```
Matrix multiplyMatrices(Matrix, Matrix);
```

Uitvoer wegschrijven

De uitvoer wegschrijven is eenvoudig:

```
void writeMatrix(const char*, Matrix*);
```

Een aantal kleine aandachtspuntjes

(1) wat doet jouw programma als de eerste regel(s) van een invoerbestand leeg is (zijn)?

(2) wanneer mag je twee matrices überhaupt met elkaar vermenivuldigen?

Segmentation faults

Je moet voor het eerst gebruik maken van pointers. Ongetwijfeld loop je een keer tegen een zogenaamde *segmentation fault* aan.[1] In deze opdracht heb je 99 van de 100 keer een array out-of-bounds laten gaan: kijk goed naar je indices. Dat wil zeggen: *print ze naar je uitvoer, en vergeet de uitvoer niet te flushen indien nodig.* Zijn ze altijd netjes binnen de grenzen?

13. Postkantoor.

Je hebt het gebracht tot directeur van een postkantoor. Soms denk je met heimwee terug aan de tijd dat je Programmeren deed en je komt op het idee om de gang van zaken in het postkantoor op je PC te simuleren. Er zijn P loketten open. In een bepaalde tijd T (bijv. één dag) komen er N klanten. Als er een klant binnenkomt, trekt hij een volgnummerje. De handelingen aan de loketten vergen verschillende tijden. Het gemiddelde van deze tijden noemen we τ. De bedoeling is om uit te vissen hoeveel mensen er gemiddeld in de rij staan en hoelang.

Omdat de komst van de klanten (naar je mag aannemen) ongecorreleerd is (d.w.z. dat de komst van een nieuwe klant niet afhangt van het feit of er kort tevoren een andere klant is geweest), zijn de mogelijke aankomsttijden volkomen random. Je begint dus met het vullen van een array (ter lengte N) met de aankomsttijden van de klanten. Vervolgens worden deze aankomsttijden gesorteerd, zodat de eerste klant op plaats 1 in de array staat. De eerste klant gaat naar loket 1.

Zodra hij hier gearriveerd is, bepaal je met behulp van een functie `GeefLoketTijd` de tijd die aan dit loket nodig is. Deze functie levert een random getal volgens de verdeling:

$$P(t) \propto te^{-t^2}.$$

Het gemiddelde van deze verdeling is $\sqrt{\pi}/2$. Je krijgt deze verdeling door een random getal x tussen 0 een 1 te transformeren volgens:

$$t = \sqrt{-\ln(x)}.$$

Dan heeft t de gewenste verdeling. Om nu een verdeling van deze vorm met een gemiddelde τ te krijgen, vermenigvuldig je t met $2\tau/\sqrt{\pi}$.

Er hebben telkens *gebeurtenissen* plaats: òf er komt een nieuwe klant binnen òf er is er een geholpen en deze vertrekt. Na elke gebeurtenis moet je nagaan wat de volgende gebeurtenis is, d.w.z. of er een klant binnenkomt of dat er een vertrekt en in dit laatste geval aan welk loket dit gebeurt. Als een klant vertrekt aan loket i, dan wordt de klant met het volgende nummertje geholpen aan loket i. Met de bovenstaande functie wordt weer een nieuw tijdstip worden gegenereerd waarop deze klant vertrekt. Op elk moment is er dus minimaal één gebeurtenis mogelijk (komst van een klant, alle loketten zijn vrij) en maximaal $P+1$ (vertrek van één van de P loketten of komst van een klant). Na elke gebeurtenis zijn er dus tussen 1 en $P+1$ nieuwe gebeurtenissen mogelijk die allen op bekende tijden zullen plaatsvinden. Welke de eerstvolgende gebeurtenis is, vind je door het minimum van deze tijden te zoeken.

Bepaal zo hoeveel klanten er gemiddeld geholpen kunnen worden. Hiervoor moet je het programma een aantal malen laten lopen. Dit laatste doe je door de hele berekening in een functie (`BepaalKlantenVerloop()`) onder te brengen die je vanuit main aanroept. Om de berekening nu een aantal keren uit te voeren plaats je de functie `BepaalKlantenVerloop()` in een loop in main.

[1] In een Windows omgeving krijg je vermoedelijk een popup met een tekst als "this program executed an illegal instruction".

14. Driehoek van Pascal

De driehoek van Pascal is een rangschikking van de binomiaalcoëfficiënten $\binom{n}{k}$ in rijen voor toenemende n beginnend met n=0 en op elke rij de n+1 binomiaalcoëfficiënten voor de mogelijke waarden van k.

Bereken de binomiaalcoëfficiënten $\binom{n}{k}$ m.b.v. de formule

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

Definiëer de faculteits functie $n!$ **recursief** en print de driehoek van Pascal voor $n = 0$ tot een variabele maximale waarde. Zorg er voor dat de driehoek *symmetrisch* t.o.v. de waarde voor $n = 0$ wordt weergegeven:

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5  10  10   5   1
        1   6  15  20  15   6   1
      1   7  21  35  35  21   7   1
    1   8  28  56  70  56  28   8   1
  1   9  36  84 126 126  84  36   9   1
1  10  45 120 210 252 210 120  45  10   1
```

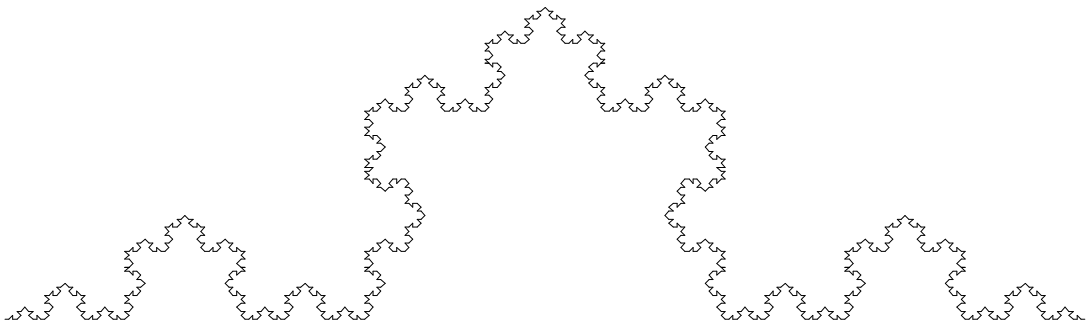
Note: In de driehoek komt de eigenschap tot uitdrukking dat elke binomiaalcoëfficiënt de som is van de twee bovenliggende. De getallen in de driehoek geven het aantal wegen aan vanaf de top naar de plaats van zo'n getal. Omdat er steeds 2 keuzen zijn om de weg naar onder te vervolgen is de som van de getallen op een rij de overeenkomstige macht van 2. Zie verder ook: http://nl.wikipedia.org/wiki/Driehoek_van_Pascal

Intermezzo

In de hierna volgende opgaven is het soms mogelijk, of vereist, om grafisch weer te geven wat er berekend wordt. Hiertoe kun je gebruik maken van de grafische bibliotheek OpenGL. Deze bibliotheek bevat een aantal functies die het grafische leven eenvoudiger maken. Een belangrijk voordeel is bijvoorbeeld dat je alle tekeningen in gebruikerscordinaten kunt maken.
Links naar beschrijvingen, voorbeelden en toepassingen van OpenGL vind je op web site van dit college: `http://mbp.science.ru.nl/leo_meerts/programmeren/`.

15. Recursie

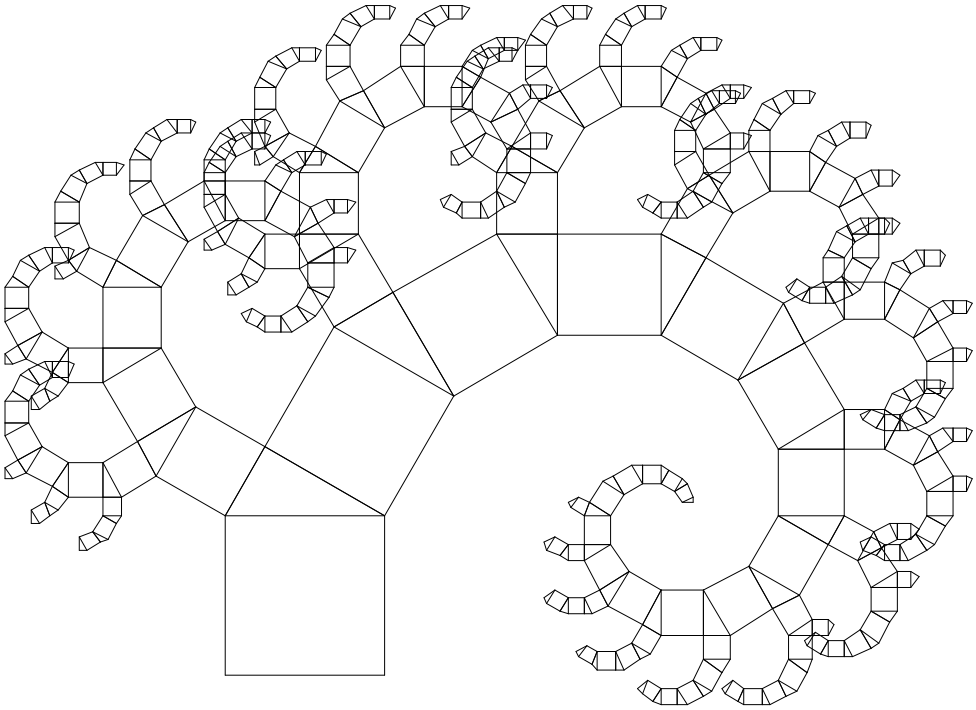
(I) Schrijf een recursieve procedure die het plaatje van figuur 1 m.b.v. OpenGL op het scherm tekent:



Figuur 1: *Koch-kromme.*

Deze "Koch-kromme" is een voorbeeld van een zogenaamde fractal. Bij elke stap wordt precies één-derde uit het midden van een lijntje weggenomen en vervangen door twee lijntjes die samen met het weggehaalde stuk een gelijkzijdige driehoek vormen. Daarna wordt voor elk van de vier resulterende lijnstukjes weer hetzelfde procédé uitgevoerd. Als afbreekvoorwaarde neem je dat de lijntjes niet onder een minimale lengte uitkomen.

(II) De zogenaamde “boom van pythagoras” is een figuur die is opgebouwd uit een “kiem”, bestaande uit een vierkantje met daarop een driehoek met hoeken van 30, 60 en 90 graden. Door deze kiem op een bepaalde manier te herhalen, wordt het plaatje van figuur 2 opgebouwd.



Figuur 2: *Boom van Pythagoras.*

Schrijf een recursief programma dat dit plaatje m.b.v. OpenGL op het scherm tekent.
Als afbreek-voorwaarde neem je dat de lengte van de zijde van het vierkant niet onder een minimale lengte komt.